# AIM 65

## MANUAL DEL LENGUAJE BASIC

I.E.P. SANT JOAN DESPI
DEPARTAMENT
ELECTRONICA

Traducción:

Bernardo Díaz
Gumersindo García
Jesús Mayor

Rockwell International

Document No 29650 N49

AIM 65

## INDICE


100 Instalación del BASIC en el AIM-65 . . . . . 100-1

Procedimiento de Instalación de la ROM . . . . . 100-1
Introducción del BASIC . . . . . 100-1
Para salir del BASIC . . . . . 100-2
Para volver a entrar al BASIC . . . . . 100-3
Cursor del BASIC . . . . . 100-3
Control de la Impresora . . . . . 100-4

200 Introducción al BASIC . . . . . 201-1

201 Juego de Comandos del BASIC . . . . . 201-1
202 Comandos Directos e Indirectos . . . . . 202-1
Comandos Directos . . . . . 202-1
Comandos Indirectos . . . . . 202-1
203 Operaciones sobre líneas y programas . . . . . 203-1
Listar un Programa . . . . . 203-1
Borrar una Línea . . . . . 203-1
Cambiar una línea . . . . . 203-1
Suprimir un Programa . . . . . 203-2
204 Impresión de Datos . . . . . 204-1
205 Formato de los Números . . . . . 205-1
206 Variables . . . . . 206-1
Asignación de Variables con una Instrucción Input . . . . 206-1
Nombres de variables . . . . . 206-1
Asignación de Variables con una Instrucción LET . . . . . 206-2
Palabras Reservadas . . . . . 206-3
Comentarios . . . . . 206-3
207 Examen de Relaciones . . . . . 207-1
Un Programa que utiliza relaciones . . . . . 207-1
208 Ciclos . . . . . 208-1
Programa para una Raíz Cuadrada . . . . . 208-1
Un Programa mejorado para la Raíz Cuadrada . . . . . 208-1
Instrucciones del BASIC para ciclos . . . . . 208-2
Otro Programa para la Raíz Cuadrada . . . . . 208-2
Programa para un Contador Decreciente . . . . . 208-3
Otras operaciones con ciclos . . . . . 208-3

209 Operaciones con Matrices . . . . . 209-1
Programa de Clasificación . . . . . 209-1

210 Subrutinas . . . . . 210-1
Terminación de un programa . . . . . 210-2

211 Introducción de Datos . . . . . 211-1

212 Cadenas de caracteres . . . . . 212-1
Función LEN . . . . . 212-1
Función LEFT$ . . . . . 212-1
Función RIGHT$ . . . . . 212-2
Función MID$ . . . . . 212-2
Concatenación de cadenas . . . . . 212-3
Funciones VAL Y STR$ . . . . . 212-4
Función CHR$ . . . . . 212-4
Ordenación de cadenas . . . . . 212-4
Otras consideraciones sobre Cadenas . . . . . 212-5

300 Definiciones de las Instrucciones . . . . . 301-1

301 Caracteres especiales . . . . . 301-1
@ . . . . . 301-1
DEL . . . . . 301-1
RETURN . . . . . 301-1
$F_1$ . . . . . 301-1
: (Dos puntos) . . . . . 301-1
2 . . . . . 301-1
$ . . . . . 301-1
% . . . . . 301-1
! . . . . . 301-2
CNTL PRINT . . . . . 301-2
302 Operadores . . . . . 302-1
Reglas para Evaluación de Expresiones . . . . . 302-1

303 Comandos . . . . . 303-1
CLEAR . . . . . 303-1
CONT . . . . . 303-1
FRE . . . . . 303-2
LIST . . . . . 303-2
LOAD . . . . . 303-3
NEW . . . . . 303-3
PEEK . . . . . 303-3
POKE . . . . . 303-3
RUN . . . . . 303-4
SAVE . . . . . 303-4

304 Instrucciones de Programa . . . . . 304-1

DEF . . . . . 304-1
DIM . . . . . 304-1





END . . . . . 304-2
FOR . . . . . 304-2
GOSUB . . . . . 304-3
GOTO . . . . . 304-4
IF . . . GOTO . . . . . 304-4
IF . . . THEN . . . . . 304-4
LET . . . . . 304-4
NEXT . . . . . 304-4
ON . . . GOSUB . . . . . 304-5
ON . . . GOTO . . . . . 304-5
REM . . . . . 304-5
RESTORE . . . . . 304-6
RETURN . . . . . 304-6
STOP . . . . . 304-6
USR . . . . . 304-6
WAIT . . . . . 304-7

305 Instrucciones de Entrada/Salida . . . . . 305-1

DATA . . . . . 305-1
INPUT . . . . . 305-1
PRINT . . . . . 305-2
READ . . . . . 305-2
SPC . . . . . 305-3
TAB . . . . . 305-3

306 Funciones de Cadena . . . . . 306-1

ASC . . . . . 306-1
CHR$ . . . . . 306-1
GET . . . . . 306-1
LEFT$ . . . . . 306-1
LEN . . . . . 306-1
MID$ . . . . . 306-1
RIGHT$ . . . . . 306-2
STR$ . . . . . 306-2
VAL . . . . . 306-2

307 Funciones Aritméticas . . . . . 307-1

ABS . . . . . 307-1
ATN . . . . . 307-1
COS . . . . . 307-1
EXP . . . . . 307-1
INT . . . . . 307-1
LOG . . . . . 307-1
RND . . . . . 307-2
SGN . . . . . 307-2

SIN . . . . . 307-2
SQR . . . . . 307-2
TAN . . . . . 307-2
Funciones Derivadas . . . . . 307-2

A Mensajes de Error . . . . . A-1
B Sugerencias sobre Espacios . . . . . B-1
Información de Asignación de Memoria . . . . . B-2
C Sugerencias sobre Velocidad . . . . . C-1
D Conversión de Programas BASIC no escritos para el AIM-65 . . . . D-1
E Código ASCII de caracteres . . . . . E-1
F Subrutinas en Lenguaje Ensamblador . . . . . F-1

La Instrucción USR(W) . . . . . F-1
Asignación de Memoria . . . . . F-1
Introducción de la Subrutina en Memoria . . . . . F-2
Introducción en Memoria de la Dirección de comienzo de la Subrutina . . . . . F-2
Un ejemplo de Instrucción USR(W) . . . . . F-3
G Grabación de Programas BASIC AIM-65 en Cassette . . . . . G-1

Grabación en Cassette mediante el comando SAVE del BASIC . . G-1
Recuperación de un Programa en Cassette mediante el Comando LOAD del BASIC . . . . . G-2
H Implementación de ATN . . . . . H-1



# INTRODUCCION

Antes de que un ordenador pueda realizar cualquier función útil, se le tiene que "decir" que hacer. Desgraciadamente, en este momento, los ordenadores no entienden Castellano o cualquier otro lenguaje "humano". La principal causa es el que nuestros lenguajes contienen muchas ambigüedades y sentidos implícitos. Al ordenador se le han de dar las instrucciones precisas y la secuencia exacta de operaciones a realizar para una tarea específica. Por esta razón se han desarrollado los lenguajes de alto nivel, con la idea de facilitar la comunicación humana con el ordenador.

El BASIC del ROCKWELL AIM-65 8 K de Microsoft es un lenguaje de programación fácil de entender y usar. Sirve como una excelente "herramienta" para aplicaciones en las áreas de gestión científica y educación. Después de pocas horas de utilización del BASIC, descubrirá que ya puede escribir programas con una facilidad que pocos lenguajes de ordenadores pueden dar.

El BASIC originalmente se desarrolló en la Universidad de Dartmouth, y ha encontrado una gran aceptación en el campo de los ordenadores. Aún siendo un lenguaje de ordenador muy simple de utilización, el BASIC es muy potente y utiliza como "comandos" un pequeño conjunto de palabras del inglés habitual. Diseñado específicamente como un lenguaje "interactivo", se puede dar un comando tal que "PRINT 2 + 2" y el BASIC responderá inmediatamente con un "4". No es necesario tener un programa en tarjetas y después esperar horas por los resultados. Tendrá también todas las posibilidades de un ordenador al alcance de sus manos.

Esperamos que disfrute con el BASIC, y que tenga éxito en la utilización y resolución de todos los problemas de programación. Para una sólida información sobre programación en BASIC, le sugerimos que use algún libro de los existentes sobre el tema.

Algunos de los mejores libros son: Dwywer, T.A. y Critchfield, M. BASIC and the Personal Computer, Addison-Wesley, Reading MA, 1978, Lenguaje BASIC de Robert L. Albrecht, Le Roy Finkel, Jerry Broown, Limusa, 1976. Programación en BASIC, Mario V. Farina, Prentice Hall, 1974.

## SECCION: INSTALACION DEL BASIC EN EL AIM–65

## TEMA:

### PROCEDIMIENTO DE INSTALACION DE LA ROM

Antes de utilizar el circuito ROM con el BASIC, asegúrese de observar las precauciones que se dan en la sección 1.4. de la guia de usario del AIM–65.

Para instalar las ROMS, apagar la alimentación del AIM–65. Inspeccionar los pins de las dos ROMS conteniendo el BASIC, comprobar que están derechas y libres de materiales extraños. Insertar la ROM número R 3225 en el zócalo Z 25, apoyando la placa base del AIM 65, por debajo del zócalo de la ROM, teniendo cuidado de que el dispositivo ROM esté con la orientación adecuada. Ahora colocar la ROM número R 3226 en el zócalo Z 26. Comprobar que ambas ROMS están completamente insertadas en sus zócalos, después conectar la alimentación del AIM–65.

### INTRODUCCION DEL BASIC

Para la introducción e inicialización del BASIC, dar un 5 por teclado una vez el monitor presente el caracter de estar dispuesto para admitir comandos. El AIM–65 responderá con:

< 5 >

MEMORY SIZE? ^

Introducir por teclado y en decimal la dirección más alta de memoria a asignar al programa BASIC. Acabar la entrada de datos con RETURN. El BASIC se asignará en la memoria, desde la posición 530 (212 en hexadecimal) a la dirección que hayamos introducido. Si el BASIC puede utilizar toda la memoria disponible, dar un RETURN sin introducir ninguna dirección. La dirección más alta disponible en el AIM 65 en la versión de 1 k de RAM es 1024 (400 en hexadecimal), y 4096 (1000 en hexadecimal) para la versión de 4 K de RAM.

ATENCION

> Todas las posiciones de memoria que se han asignado al BASIC son inicializadas con el valor hexadecimal AA. Esta inicialización, por supuesto, destruye toda la información que hubiera sido almacenada anteriormente en esta zona de memoria. Por esta razón, debe salvar cualquier dato o programa de importancia que esté en este área antes de introducir el BASIC con la tecla 5.

No obstante el BASIC puede coexistir en memoria con el texto del buffer de texto o con una rutina en lenguaje máquina, con tal de que estos ocupen posiciones de

memoria mas altas que la que demos como dirección superior en la inicialización del BASIC.

Después el BASIC preguntará:

WIDTH? ∧

Introducir por teclado el número de caracteres por línea de la impresora de salida (o de cualquier otro dispositivo que se esté utilizando como salida) y finalizar la entrada con un RETURN.

El número introducido puede variar desde 1 a 255, dependiendo del dispositivo de salida. Si damos al RETURN sin introducir ningún número, se tomará por defecto el valor 20, que es el número de caracteres por línea de la impresora del AIM–65.

El BASIC responderá con:

XXXX BYTES FREE

Donde XXXX es el número de bytes disponibles para el programa BASIC, variables, matrices y cadenas. Si se asignó toda la memoria existente, el BASIC responderá con:

494 BYTES FREE (para la versión de 1 K de RAM; esto es 1024–530)
ó
3566 BYTES FREE (para la versión de 4 K de RAM; esto es 4096–530).

El BASIC presentará:

∧AIM 65 BASIC Vn. n
donde n.n, es el número de versión.

El BASIC está ya disponible, el carácter ( ∧ ) presentado en la columna 1 indica que se puede comenzar la entrada de comandos. El tema 201 le permitirá comenzar a introducir los comandos BASIC.

Sin embargo, lea los siguientes párrafos para comprender como salir y volver a entrar al BASIC, así como el funcionamiento del cursor del BASIC.

PARA SALIR DEL BASIC

Para salir del BASIC y volver al Monitor del AIM 65, teclear ESC cuando esté presente el cursor de comandos BASIC. Puede también salir del BASIC durante la ejecución de un programa, pulsando la tecla F1 (ver tema 301). Después de pulsar la tecla F1,





tiene dos opciones: pulsar el ESC para volver al Monitor del AIM–65, o introducir un nuevo comando BASIC.

Pulsando el interruptor de RESET también se fuerza a la introducción del Monitor del AIM–65, realizándose un reset por hardware.

PARA VOLVER A ENTRAR AL BASIC

Se puede volver a entrar al BASIC tecleando un 6 en cualquier momento que el Monitor del AIM–65 presente el carácter de aviso ( ). Sin embargo, en este caso no se destruye el programa BASIC que esté en memoria. El AIM–65 responderá a la tecla 6 con:

< 6 >

<6>

CURSOR DEL BASIC

Cuando aparece en la columna 1 el cursor ( ∧ ), indica que se pueden introducir los comandos del BASIC. En el display permanecen los datos que resultan de la ejecución del último comando introducido, excepto la columna 1, para que exista continuidad entre el comando anterior o los datos presentados y lo siguiente, esto es muy útil cuando la impresora está desconectada para ahorrar papel.

Cuando se teclea el primer carácter del siguiente comando, se limpia el display, quedando solo el nuevo carácter introducido. El cursor avanzará por el display de acuerdo con los caracteres tecleados para indicar la posición del carácter a introducir.

El cursor presentado en el display no aparece sobre la impresora, por tanto en ella los datos de la columna 1 permanecen inalterados.

ATENCION

En el modo de entrada de comandos BASIC, el signo menos (–) de cualquier valor negativo que al ser presentado aparezca en la columna 1, será reemplazado por el cursor del BASIC. En comandos directos el signo menos será un destello antes de que se presente el cursor. La impresora lo imprimirá si está activa, sino no será perceptible. Para que aparezca colóquese en la instrucción un espacio en blanco delante del signo. (Ver tema 204).

## SECCION: INSTALACION DEL BASIC EN EL AIM–65

### TEMA

CONTROL DE LA IMPRESORA

Mientras estamos en modo de introducción de comandos BASIC, la impresora se activa o desactiva pulsando simultáneamente las teclas CNTL y PRINT y a continuación aparece en el display el estado de la impresora (ON/OFF).

Si la impresora se desactiva, los comandos de entrada, las instrucciones y los datos de salida referenciados en los comandos PRINT, serán dirigidos únicamente al display. Si la impresora está activa los comandos y datos referenciados en las instrucciones PRINT serán dirigidos a la impresora y al display.

Con la impresora desactivada los datos también pueden dirigirse a la impresora utilizando el comando PRINT!. (Ver tema 305).

Idénticamente sucede con las instrucciones INPUT que reproducirán los datos por la impresora según el estado de control de ésta. Una instrucción INPUT! reproducirá los datos por la impresora aunque esté desactivada. (Ver tema 305).



## SECCION: INTRODUCCION AL BASIC

### TEMA : JUEGO DE COMANDOS DEL BASIC

Esta sección no pretende ser un curso detallado de programación en BASIC. Sin embargo, servirá como una excelente introducción para aquéllos que no estén familiarizados con el lenguaje.

Le recomendamos que pruebe cada ejemplo de esta sección tal como se presenta. Esto aumentará su interés por el BASIC y por su utilización. La Tabla 201–1 da una lista de todos los comandos del BASIC del AIM–65 y la referencia del tema donde se explica cada tipo de comando.

NOTA

En cualquier momento en que el cursor aparece en la columna 1, se puede introducir un comando BASIC. Todos los comandos BASIC acaban cuando se pulsa el RETURN. El RETURN dice al BASIC que ya ha finalizado de introducir el comando. Si comete un error, teclee del (RUBOUT en el TTY) para eliminar el último carácter. Repita el uso del DEL para eliminación sucesiva de caracteres. Pulsando ⌀ eliminará la línea que se está introduciendo.

Tabla 201–1 Comandos del BASIC del AIM–65

| Comandos (Tema 303) | | | |
|---|---|---|---|
| | | IF . . . GOTO | |
| | | IF . . . THEN | |
| CLEAR | | LET | |
| CONT | | NEXT | |
| FRE | | ON . . . GOSUB | |
| LIST | | ON . . . GOTO | |
| LOAD | | REM | |
| NEW | | RESTORE | |
| PEEK | | RETURN | |
| POKE | | STOP | |
| RUN | | USR | |
| SAVE | | WAIT | |
| Instrucciones de Programa (Tema 304) | | Entrada/Salida (Tema 305) | |
| | | DATA | |
| DEF FN | | GET | |
| DIM | | INPUT | |
| END | | READ | |
| FOR | | PRINT | |
| GOSUB | | SPC | |
| GOTO | | TAB | |

| Funciones con Cadenas de Caracteres (Tema 306) | Funciones aritméticas (Tema 307) |
|---|---|
| ASC | ABS |
| CHR$ | * ATN |
| LEFT$ | COS |
| LEN | EXP |
| MID$ | INT |
| RIGHT$ | LOG |
| STR $ | RND |
| VAL | SIN |
| | SGN |
| | SQR |
| | TAN |

* Aunque la función ATN no esté incluida en el BASIC del AIM–65, el comando ATN es reconocido (Ver apéndice H).





COMANDOS DIRECTOS

Pruebe a introducir por teclado lo siguiente:

PRINT 10–4 (Finalizar con RETURN)

El BASIC inmediatamente escribirá:

6

La instrucción de imprimir que tecleó fue ejecutada tan pronto como pulsó la tecla RETURN. Esto se conoce por el nombre de comando directo. El BASIC evalua la fórmula que hay detrás del PRINT y luego saca el valor, en este caso 6.

Ahora probar a introducir:

PRINT 1/2, 3 *10 ("*" significa multiplicar, "/" dividir)

El BASIC escribirá:

.5 30

Como se ve, con el BASIC se pueden efectuar operaciones como sumar, restar, multiplicar o dividir. Observar que la coma "," se utilizó con el comando de impresión para escribir los dos valores uno junto al otro. El comando divide la línea en columnas de 10 caracteres de ancho. El comando BASIC provoca un salto al campo siguiente del terminal, donde se escribe el valor 30.

COMANDOS INDIRECTOS

Existe otro tipo de comandos conocidos por el nombre de Comandos Indirectos. Cada comando indirecto comienza con un Número de Línea. Un Número de Línea es cualquier entero entre 0 y 63999.

Intente introducir estas líneas:

```
10 PRINT 2+3
20 PRINT 2–3
```

Una secuencia de comandos indirectos se llama "Programa". El BASIC guarda en memoria los Comandos Indirectos, en lugar de ejecutarlos inmediatamente. Cuando teclee RUN, el BASIC ejecutará las instrucciones indirectas que tenemos en memoria,

comenzando en primer lugar por la que tiene un número de línea más bajo, después irá al próximo más alto, etc., hasta acabar con todas las que hemos introducido.

En el ejemplo anterior, hemos puesto a la primera línea el número 10 y a la segunda el 20. Sin embargo no importa el orden en que hayamos introducido las instrucciones indirectas. El BASIC siempre las coloca en el orden numérico correcto en función del Número de Línea.

Suponga que tecleamos

```
RUN
```

El BASIC escribirá

```
 5
-1
```





En el tema 202, introdujimos un programa de dos líneas en memoria. Ahora vamos a ver como se puede usar el BASIC para trabajar sobre una o ambas líneas del programa.

LISTAR UN PROGRAMA

Si queremos listar completamente un programa que está en memoria, tecleamos

```
LIST
```

El BASIC responderá con:

```
10 PRINT 2+3
20 PRINT 2-3
```

BORRAR UNA LINEA

En ocasiones se desea suprimir completamente una Línea del programa. Esto se efectúa dando el Número de Línea a suprimir seguido por un RETURN.

Teclear lo siguiente:

```
10
LIST
```

El BASIC responderá

```
20 PRINT 2-3
```

Hemos suprimido la línea 10 del programa

CAMBIAR UNA LINEA

Se puede cambiar la línea 10, antes de borrarla, dando una nueva línea 10 y acabando con RETURN.

Teclear lo siguiente:

```
10 PRINT 3-3
LIST
```

El BASIC responderá con:

```
10 PRINT 3-3
20 PRINT 2-3
```

No es recomendable que la numeración de las líneas sea consecutiva, ya que puede ser necesario insertar una nueva línea entre dos existentes. Un incremento de 10 en los números de línea de dos líneas consecutivas suele ser suficiente.

SUPRIMIR UN PROGRAMA

Si desea suprimir completamente un programa guardado en memoria, introduzca por teclado "NEW". Si terminó de ejecutar un programa y desea leer uno nuevo, asegúrese de introducir primero "NEW".

Introduzca lo siguiente:

```
NEW
```

Compruebe que se suprimió el programa, no obtendrá listado:

```
LIST
```





Es frecuente incluir textos que expliquen las respuestas escritas.

Introducir por el teclado lo siguiente:

```
PRINT "UN MEDIO IGUAL", 1/2
```

El BASIC responderá con:

```
UN MEDIO IGUAL
 .5
```

Como se explicó en el tema 202, al incluir una "," en la instrucción PRINT obligamos a dividir el espacio de impresión en campos de 10 columnas antes de que se escriba el valor que sigue a la coma.

Si utilizamos un ";" en lugar de la coma, el siguiente valor se escribirá inmediatamente después del valor anterior.

NOTA

> Los números siempre se escriben al menos con un blanco de separación. Cualquier texto debe estar entre comillas.

Probar los siguientes ejemplos:

1. 
```
PRINT "UN MEDIO IGUAL A"; 1/2
UN MEDIO IGUAL A  .5
```

2. 
```
PRINT 1, 2, 3
1            2
3
...
```

3. 
```
PRINT 1; 2; 3
1 2 3
```

4. 
```
PRINT -1; 2;-3
-1 2 -3
```

Por un momento haremos una pequeña disgresión para explicar el formato de los números en BASIC. Los números se almacenan internamente con nueve dígitos de precisión. Cuando se escribe un número, sólo se muestran nueve dígitos. Cada número puede tener un exponente (un factor de escala en potencias de diez).

El mayor número que se puede presentar con el BASIC del AIM 65 es $1.70141183_{*}10^{38}$, mientras que el número positivo más pequeño es $2.93873588_{*}10^{-39}$.

Las siguientes reglas definen el formato de impresión de un número:

1. Si el número es negativo, se escribe un signo (−). Si el número es positivo, se dá un espacio.

2. Si el valor absoluto del número es un entero comprendido entre 0 y 999999999, se escribe como un entero.

3. Si el valor absoluto del número es mayor o igual a 0.01 y menor o igual a 999999999, el valor es escrito con notación en coma fija, sin exponente.

4. Si el número no está incluido en las categorías 2 ó 3, se utiliza la notación científica.

El formato de la notación científica es el siguiente:
SX.XXXXXXXXESTT. (Cada x indica un valor entero cualquiera del 0 al 9).

La "S" de la posición de comienzo es el signo del número: un espacio indica un número positivo, y un signo "−" indica un número negativo. Delante del punto decimal se escribirá un dígito distinto de cero. Este será seguido por un punto decimal y después de ocho dígitos de la mantisa. Luego se escribirá una "E" (para el exponente), seguida por el signo del exponente, luego los dos dígitos TT son el exponente. Nunca se escriben los ceros en las posiciones más a la izquierda y antes de las cifras significativas, por ejemplo: el dígito anterior al punto decimal nunca es cero. Intente escribir ceros, no lo conseguirá.

Si existe un sólo dígito después de varios ceros, estos son suprimidos durante la impresión y no se escribirá el punto decimal. Los signos del exponente serán " + " para el positivo y "−" para el negativo. Siempre se escriben dos dígitos para el exponente, es decir los ceros no se suprimen en el campo del exponente. El valor de cualquier número expresado de esta forma, es el número a la izquierda de la "E" multiplicado por la potencia de 10 que indique el número a la derecha de la "E".

Sin importar el tipo de formato utilizado, siguiendo al número, siempre se escribe un espacio. El BASIC comprueba si el número cabe en la línea en curso. Si no cabe, se produce un salto de línea antes de imprimir el número.





A continuación vemos ejemplos de varios números y sus formatos de salida, tal como aparecen en BASIC:

| NUMERO | FORMATO DE SALIDA |
|---|---|
| 1 | 1 |
| −1 | −1 |
| 6523 | 6523 |
| −23.460 | −23,46 |
| 1 E 20 | 1 E + 20 |
| −12.3456E−7 | −1.23456E−06 |
| 1.234567E−10 | 1.234567E−10 |
| 1000000000. | 1E + 09 |
| 999999999 | 999999999 |
| .1 | .1 |
| .01 | .01 |
| .000123 | 1.23E−04 |

Una entrada de un número desde el teclado o una constante numérica para un programa en BASIC puede tener tantos dígitos como se desee, hasta la longitud de la línea (72 caracteres) o hasta el valor numérico máximo. Sin embargo, sólo los 10 primeros dígitos son significativos, y el décimo es redondeado por exceso.

```
PRINT 1.23456789876543210
1.2345679
```

ASIGNACION DE VARIABLES CON UNA INSTRUCCION INPUT

El siguiente programa es un ejemplo que lee un valor del teclado, este valor se utiliza para calcular un resultado e imprimirlo.

```
10      INPUT R
20      PRINT 3.14159*R*R
RUN
? 10
314.159
```

Lo que ha pasado es lo siguiente:
Cuando el BASIC encontró la instrucción de entrada, presentó por el display como señal una interrogación (?) y luego esperó que se le diese un número por teclado. Cuando se hizo (en el ejemplo anterior tecleamos 10), la ejecución continuó con la siguiente instrucción del programa después de asignar a la variable (R) el valor introducido (en este caso 10). Ahora se ejecutaría la línea 20. Cuando se evalua la fórmula de la instrucción PRINT, cada vez que aparece la variable (R) es sustituida por su valor (10). Por tanto la fórmula queda como 3.14159*10*10, o 314.159.

Si deseamos calcular el área de varios círculos, volveríamos a ejecutar el programa para los sucesivos círculos. Pero hay un camino para facilitar el trabajo, que consiste en añadir una línea al programa, del siguiente modo:

```
30 GOTO 10
RUN
?10
 314.159
? 3
 28.27431
? 4.7
 69.3977231
?
```

Poniendo la instrucción GOTO al final de nuestro programa, hemos obligado a que vuelva a la línea 10 después de imprimir la respuesta para los distintos círculos. Esto se realizaría indefinidamente, pero decidimos parar después de calcular el área de 3 círculos. Para parar tecleamos RETURN como respuesta a la siguiente instrucción INPUT (es decir una línea en blanco).

NOMBRES DE VARIABLES

La letra "R" en el programa anterior es una "variable". Un nombre de variable puede ser cualquier carácter alfabético y puede estar seguido por cualquier carácter alfanumérico (letras A a la Z, número del 0 al 9).



SECCION: INTRODUCCION AL BASIC

TEMA: VARIABLES

Cualquier carácter alfanumérico después de los dos primeros es ignorado.

Aquí tenemos algunos ejemplos de nombres de variables legales e ilegales:

| LEGAL | ILEGAL | |
|---|---|---|
| A | % | (el primer carácter debe ser alfabético) |
| Z1 | ZIABCD | (Nombre de variable demasiado largo) |
| TP | TO | (Las palabras reservadas no pueden ser nombres de variables). |
| PSTG $ | TGOTO | (Los nombres de variables no pueden contener palabras reservadas). |
| COUNT | | |

DANDO VALORES A LAS VARIABLES CON UNA INSTRUCCION LET O UNA ASIGNACION.

Además de asignar valores a las variables con una instrucción input, también se puede hacer con una instrucción LET o sentencia de asignación.

Pruebe los siguientes ejemplos:

```
A = 5
PRINT A,A*2
 5    10
LET Z = 7
PRINT Z, Z-A
 7     2
```

Como puede apreciar en los ejemplos, la palabra "LET" es opcional en la instrucción de asignación.

El BASIC "recuerda" los valores que han sido asignados a las variables usando este tipo de instrucción. Este proceso de"recuerdo" necesita de un espacio en memoria para almacenar los datos.

Los valores de las variables se pierden y el espacio utilizado en memoria para almacenarlas se libera cuando ocurre una de estas cuatro condiciones:

- Se introduce una nueva línea en el programa o se borra una vieja.
- Se introduce un comando CLEAR por el teclado.
- Se dá un comando RUN.
- Se introduce un NEW.

Otro hecho importante es que si se encuentra una variable en una fórmula antes de asignarle un valor, se le asigna el valor cero automáticamente. El cero se toma como valor de la variable en esa fórmula particular. Probar el ejemplo siguiente:

```
PRINT Q ; Q + 2; Q * 2
0 2 0
```

PALABRAS RESERVADAS

Las palabras utilizadas como instrucciones BASIC, están reservadas para ese propósito específico. No se pueden utilizar estas palabras como nombres de variables o como parte de un nombre de variable. Por ejemplo "FEND" sería ilegal puesto que "END" es una palabra reservada.

La tabla 206–1, es una lista de las palabras reservadas en BASIC.

Tabla 206 Palabras Reservadas del BASIC del AIM–65

| | | | | |
|---|---|---|---|---|
| ABS | FN | LIST | PRINT | SPC |
| AND | FOR | LOAD | POS | SQR |
| ASC | FRE | LOG | READ | STEP |
| ATN | GET | MID$ | REM | STOP |
| CHR$ | GOSUB | NEW | RESTORE | STR |
| CLEAR | GOTO | NEXT | RETURN | TAB |
| CONT | IF | NOT | RIGHT$ | TAN |
| COS | INPUT | NULL | RND | THEN |
| DATA | INT | ON | RUN | TO |
| DEF | LEFT$ | OR | SAVE | USR |
| DIM | LEN | PEEK | SGN | VAL |
| END | LET | POKE | SIN | WAIT |
| EXP | | | | |

COMENTARIOS

La instrucción REM abreviatura inglesa de "comentarios" se utiliza para insertar los comentarios o notas en un programa. Cuando el BASIC encuentra una instrucción REM, ignora el resto de la línea.

Esto sirve principalmente como una ayuda al programador y no realiza ninguna función en la operación del programa durante la solución de un problema particular.





Supongamos que deseamos escribir un programa para comprobar si un número es cero. Con las instrucciones que hemos visto hasta el momento, no lo podemos hacer. Se necesita una instrucción que nos permita bifurcar condicionalmente a otra instrucción. La instrucción "IF – THEN" hace justamente esto.

Introducir el siguiente programa. (Recuerde, introducir en primer lugar NEW).

```
10 INPUT B
20 IF B = 0 THEN 50
30 PRINT "NO ES CERO"
40 GOTO 10
50 PRINT "CERO"
60 GOTO 10
```

Ahora ejecutarlo, nos pedirá el valor de B. Introduzca cualquier valor que desee. El AIM–65 irá después a la instrucción "IF". En esta instrucción entre el IF y el THEN hay dos expresiones separadas por una "relación".

La relación será uno de los siguientes símbolos:

| RELACION | SIGNIFICADO |
|---|---|
| = | Igual a |
| > | Mayor que |
| < | Menor que |
| < > | Distinto |
| < = o = < | Menor o igual que |
| = > o > = | Mayor o igual que |

La instrucción IF es verdadera o falsa, dependiendo de que la identidad se verifique. Por ejemplo, en el programa anterior, si para B introducimos el valor cero, la instrucción IF sería verdadera puesto que 0 = 0. En este caso, como el número que sigue al THEN es 50, la ejecución del programa continuará en la línea 50. Por tanto se escribirá "CERO" y después el programa volverá a la línea 10 (debido a la instrucción GOTO de la línea 60).

Supongamos que le diésemos a B el valor 1. Cómo 1 = 0 no es cierto, la instrucción IF sería falsa y el programa continuaría su ejecución en la siguiente línea. Por tanto escribirá "NO ES CERO" y con el GOTO de la línea 40 volveríamos a la línea 10.

UN PROGRAMA QUE UTILIZA RELACIONES

Ahora vamos a probar el siguiente programa que compara dos números:

## SECCION: INTRODUCCION AL BASIC

## TEMA: EXAMEN DE RELACIONES

```
10   INPUT A, B
20   IF A< =B THEN 50
30   PRINT "A ES MAYOR"
40   GOTO 10
50   IF A<B  THEN 80
60   PRINT "SON IGUALES"
70   GOTO 10
80   PRINT "B ES MAYOR"
90   GOTO 10
```

Cuando se ejecuta el programa, en la línea 10 se introducirán dos números desde el teclado. En la línea 20 sí a es mayor que B, A< = B será falso. Esto obligará que se ejecute la siguiente instrucción, escribiendo "A ES MAYOR" y después en la línea 40 volvemos a la línea 10 para comenzar otra vez.

En la línea 20, si A tiene el mismo valor que B, A<=B es cierta, así que iremos a la línea 50. En la línea 50, como A tiene el mismo valor que B, A < B es falsa por tanto, iremos a la siguiente instrucción y escribiremos "SON IGUALES". Después la línea 70 nos envía otra vez al comienzo.

En la línea 20, si A es más pequeño que B, A<=B es cierta e iremos a la línea 50. En la línea 50, A < B será cierto y saltaremos a la línea 80. Se escribirá "B ES MAYOR" y volveremos nuevamente al comienzo.

Intentar ejecutar los dos últimos programas varias veces. Será más fácil de entender si usted escribe su propio programa utilizando la instrucción IF–THEN. Ahora el camino más fácil para comprender como trabaja el BASIC es realizar sus propios programas. Recuerde que estos programas paran su ejecución dando un RETURN como respuesta a un INPUT.



## SECCION: INTRODUCCION AL BASIC

## TEMA: CICLOS

Una ventaja de los ordenadores es su capacidad para realizar tareas repetitivas. Vamos a acercarnos al tema y veamos como trabaja.

PROGRAMA PARA UNA RAIZ CUADRADA

Supongamos que queremos una tabla con las raíces cuadradas de los números del 1 al 9. La función en BASIC para la raíz cuadrada es "SQR"; la forma es SQR (X); donde X es el número del cual deseamos la raíz cuadrada. Podríamos escribir el siguiente programa:

```
10   PRINT 1; SQR(1)
20   PRINT 2; SQR(2)
30   PRINT 3; SQR(3)
40   PRINT 4; SQR(4)
50   PRINT 5; SQR(5)
60   PRINT 6; SQR(6)
70   PRINT 7; SQR(7)
80   PRINT 8; SQR(8)
90   PRINT 9; SQR(9)
```

PROGRAMA MEJORADO PARA LA RAIZ CUADRADA

El programa anterior haría lo que deseamos, pero es ineficiente. Podemos mejorar un programa con la instrucción IF que lo optimiza considerablemente. El programa es el siguiente:

```
10   N = 1
20   PRINT N; SQR (N)
30   N=N + 1
40   IF N<= 9  THEN 20
```

Cuando el programa se ejecuta, veremos que su salida coincide exactamente con la del anterior. Vamos a ver como funciona:

En la línea 10 tenemos una instrucción LET, que asigna a la variable N el valor uno. En la línea 20 imprimimos el valor en curso de N y de su raíz cuadrada; así en nuestro caso tendríamos 20 PRINT 1; SQR (1) y la impresión del cálculo.

En la línea 30 utilizamos una instrucción que en principio puede parecer un LET poco corriente. Matemáticamente la instrucción N = N + 1 es una igualdad sin sentido. Sin embargo es importante recordar que en una instrucción LET, el símbolo " = " no significa igualdad sino asignación. Esta instrucción añade uno al valor actual de N. Por tanto, después de pasar por primera vez por la línea 30, N tendrá el valor 2.

En la línea 40, como N es igual a 2, N<=9 es cierta así que la parte THEN de la instrucción nos envía a la línea 20 con el valor de 2 para N.

En definitiva, el resultado en las líneas 20 a 40 se repiten añadiendo cada vez 1 al valor de N. Cuando N sea igual a 9 en la línea 29, en la siguiente línea se incrementará a 10. Este resultado hace que la condición de la línea 40 sea falsa, y como no hay más instrucciones el programa parará.

INSTRUCCIONES DEL BASIC PARA CICLOS

Esta técnica es conocida como "ciclo" o "iteración". Puesto que es bastante utilizada en programación, hay instrucciones especiales en BASIC para su utilización. Podemos mostrar esto en el siguiente programa:

```
10    FOR N=1 TO 9
20    PRINT N; SQR (N)
30    NEXT N
```

La salida del programa listado anteriormente será idéntica a la de los otros dos programas previos.

En la línea 10, N es igual a 1. La línea 20 escribe el valor de N y su raíz cuadrada. En la línea 30 vemos un nuevo tipo de instrucción. La instrucción "NEXT N" hace que se añada 1 a N, y luego compruebe si N<=9 para ir en el caso "cierto" a la siguiente instrucción a FOR. Este funcionamiento es idéntico al de los programas anteriores.

Notar que la variable que sigue a la palabra reservada "FOR" es la misma que la que sigue a la palabra "NEXT". La N no es ninguna variable especial. Cualquier variable podría ser utilizada en el ciclo, siempre que fuese la misma a lo largo de todas las instrucciones entre el "FOR" y el "NEXT". Por ejemplo, podríamos substituir "N" por "Z1" en el programa anterior y nos funcionaría exactamente igual.

OTRO PROGRAMA PARA LA RAIZ CUADRADA

Supongamos que deseamos construir una tabla con las raíces cuadradas de los números pares comprendidos entre 10 y 20. El siguiente programa realiza esta tarea:

```
10    N=10
20    PRINT N; SQR (N)
30    N=N+2
40    IF N<=20 THEN 20
```





Notar la similitud entre este programa y el programa mejorado de la raíz cuadrada. También puede escribirse este programa utilizando el ciclo "FOR" que acabamos de explicar.

```
10    FOR N = 10 TO 20 STEP 2
20    PRINT N; SQR (N)
30    NEXT N
```

Observe que la única diferencia entre este programa y el anterior es la adición en el ciclo "FOR" de la cláusula "STEP 2".

Esto le dice al BASIC que añada 2 a N cada vez, en lugar de uno como en el programa anterior. Si no damos la opción "STEP" en la instrucción "FOR", el BASIC asume que se debe añadir 1. El "STEP" puede ir seguido de cualquier expresión.

PROGRAMA PARA UN CONTADOR DECRECIENTE

Supongamos que queremos un contador decreciente de 10 a 1. Un programa para hacer esto sería el siguiente:

```
10    I = 10
20    PRINT I
30    I = I - 1
40    IF I>=1 THEN 20
```

Observe que ahora preguntamos si I es mayor o igual que el valor final. La razón es que ahora la cuenta la hacemos restando. En los ejemplos anteriores era lo contrario, así que comprobamos si la variable era menor o igual que el valor final.

OTRAS OPERACIONES CON CICLOS

La instrucción mostrada anteriormente, "STEP" puede utilizarse con números negativos para llegar al mismo resultado. Esto se puede hacer usando un formato similar al del otro programa.

```
10    FOR I=10 TO 1 STEP - 1
20    PRINT I
30    NEXT I
```

Los ciclos "FOR" pueden estar "anidados". Por ejemplo:

```
10    FOR I = 1 TO 5
20    FOR J = 1 TO 3
```

```
30    PRINT I, J
40    NEXT J
50    NEXT I
```

Notar que "NEXT J" precede a "NEXT I". Esto es debido a que el ciclo J está dentro del ciclo I. El siguiente programa es incorrecto; ejecutarlo y ver que sucede.

```
10    FOR I = 1 TO 5
20    FOR J = 1 TO 3
30    PRINT I, J
40    NEXT I
50    NEXT J
```

No funciona porque cuando se encuentra con "NEXT I", todos los valores del ciclo J se pierden. Esto ocurre porque los ciclos I y J están yuxtapuestos.





Frecuentemente es útil poder seleccionar cualquier elementos en una tabla de números. El BASIC permite hacer esto usando matrices.
Una matriz es una tabla de números. El nombre de esta tabla (nombre de la matriz) es cualquier nombre de variable que sea legal, por ejemplo "A". El nombre de la matriz "A" es distinto e independiente del de una variable normal "A", y se podrían utilizar ambos en un mismo programa.

Para seleccionar el elemento de una tabla, indexamos "A": Esto es, para seleccionar el elemento I, encerramos I en un paréntesis "(I)" que sigue a la "A". Por tanto "A(I)" es elemento I en la matriz "A".
"A(I)" es sólo un elemento de la matriz A, y se debe decir al BASIC cuanto espacio necesitamos para colocar en memoria la matriz entera. Esto se hace con una instrucción "DIM", utilizando el formato "DIM A (15)". En este caso, hemos reservado espacio para los índices de la matriz "I" desde el 0 al 15. Los índices de las matrices siempre comienzas en cero; por tanto, en el ejemplo anterior tenemos 16 números en la matriz A.
Si "A (I)" se utiliza en un programa antes de dimensionarla, el BASIC reserva 11 espacios para elementos (0 al 10).

PROGRAMA DE CLASIFICACION

Como ejemplo de utilización de matrices, intente clasificar una lista de 8 números con el siguiente programa, al cual le tenemos que dar los números a clasificar.

```
10    DIM A (8)
20    FOR I = 1 TO 8
30    INPUT A (I)
50    NEXT I
70    F = O
80    FOR I = 1  TO  7
90    IFA (I) <=A(I+1) THEN 140
100   T = A (I)
110   A (I) = A (I + 1)
120   A (I+1) = T
130   F = 1
140   NEXT I
150   IF F = 1 THEN 70
160   FOR I=1  TO 8
170   PRINT A (I)
180   NEXT I
```

Cuando la línea 10 es ejecutada, el BASIC dá un espacio contiguo para los 9 valores numéricos, A(0) al A(8). Las líneas de la 20 a la 50 obtienen la lista de datos del usario sin clasificar. La clasificación se realiza viendo toda la lista de números de dos en dos y cambiando la posición de todos aquellos que no estén en orden. "F" se utiliza para indicar si se ha producido un cambio en la secuencia de números; si se produjo cualquier cambio, la línea 150 dice al BASIC que vuelva y compruebe la secuencia una vez más.

Si no hubo ningún cambio de posición de los números, o ya están todos ordenados, las líneas 160 a la 180 escribirán la lista de números clasificados. Notar que los índices pueden ser cualquier expresión.

Si usted tiene un programa que realiza las mismas operaciones en distintos lugares, repetirá las mismas instrucciones para realizar las mismas operaciones en cada lugar del programa donde las necesite.

Las instrucciones "GOSUB" y "RETURN" pueden utilizarse para evitar esta duplicación. Cuando se encuentra un "GOSUB", el BASIC bifurca a la línea cuyo número sigue al "GOSUB". Sin embargo, el BASIC recuerda donde estaba en el programa antes de bifurcar. Cuando se "encuentra" la instrucción "RETURN", el BASIC vuelve a la siguiente instrucción al último "GOSUB" ejecutado. Observe el siguiente programa:

```
 10  PRINT "QUE NUMERO"
 30  GOSUB 100
 40  T=N
 50  PRINT "SEGUNDO NUMERO";
 70  GOSUB 100
 80  PRINT "LA SUMA ES";T+N
 90  STOP
100  INPUT N
110  IF N = INT (N) THEN 140
120  PRINT "DEBE SER ENTERO"
130  GOTO 100
140  RETURN
```

Este programa pide dos números (que deben ser enteros) y luego escribe su suma. La subrutina en este programa, está en las líneas 100 a la 140. La subrutina pide un número, y si no es entero, pide un nuevo número. Continuará pidiendo números hasta que le introduzcamos un valor entero.

El programa principal imprime "QUE NUMERO" y luego llama a la subrutina para obtener el valor del número N. Cuando la subrutina vuelve (a la línea 40), el valor introducido se guarda en la variable T. Esto se hace para que, cuando llamemos a la subrutina por segunda vez en busca del segundo sumando, no perdamos el primero.

Ahora escribe "SEGUNDO NUMERO" y llama a la subrutina por segunda vez, por lo que podemos introducir el segundo sumando.

Cuando volvemos de la subrutina por segunda vez se escribe "LA SUMA ES", seguido por la suma. T contiene el valor del primer sumando introducido y N el del segundo.



SECCION: INTRODUCCION AL BASIC

TEMA: SUBRUTINAS

TERMINACION DE UN PROGRAMA

La siguiente instrucción en el programa es un STOP. Esto obliga a que el programa pare en la ejecución de la línea 90. Si la instrucción "STOP" fuese excluida del programa, entraríamos en la subrutina de la línea 100. Esto no debe ocurrir puesto que pediría otro número. Si le diésemos el número la subrutina intentaría volver a la siguiente instrucción del último "GOSUB" ejecutado y como no hubo ninguno que llamase a la subrutina ocurrirá un error RG. Para cada "GOSUB" que se ejecute debe haber un "RETURN", y viceversa: sólo debe aparecer un "RETURN" cuando se llame a una subrutina con "GOSUB"

Se puede utilizar "STOP" o "END" para separar un programa de las subrutinas. Con el "STOP" se imprimirá un mensaje diciendo que se encontró una línea con un "STOP".

Supongamos que tiene que introducir números en su programa que no cambian cada vez que el programa es ejecutado, pero que por otro lado deben ser fácilmente modificables si es necesario. El BASIC contiene instrucciones especiales para este propósito, éstas son "READ" y "DATA".

Considerar el siguiente programa:

```
 10  PRINT "ADIVINE UN NUMERO"
 20  INPUT G
 30  READ D
 40  IF D = -999999 THEN 90
 50  IF D<>G THEN 30
 60  PRINT "ACERTO"
 70  END
 90  PRINT "MALA ELECCION, PRUEBE OTRA
     VEZ"
 95  RESTORE
100  GOTO 10
110  DATA 1, 393, -39, 28, 391, -8, 0, 3.14, 90
120  DATA 89, 5, 10, 15, -34, -999999
```

Cuando se encuentra la instrucción "READ" el efecto es el mismo que con una instrucción "INPUT". Pero, en vez de tomar el número del teclado, se toma el número de las instrucciones "DATA".

La primera vez que se necesita un número para un READ, se le dá el primer número que aparece en la instrucción DATA. La segunda vez que se precisa un número, se dará el segundo. Cuando todos los números del primer DATA han sido leidos de esta forma, se utilizará la segunda instrucción DATA. Los DATA son leidos siempre secuencialmente, y puede haber cualquier número de instrucciones DATA en un programa.

El propósito de este programa es hacer un pequeño juego, en el que se intenta adivinar uno de los números contenidos en las instrucciones DATA. Por cada número introducido, se leen todos los números contenidos en las instrucciones DATA, hasta que se encuentra el número o se acaban los valores.

Si leemos más valores que números hay en las instrucciones DATA, ocurre un error de salida de datos (OD). Por esto en la línea 40 comprobamos si el valor leido es -999999. Este no es un número del juego, se utiliza como indicador de que todos los datos han sido leidos (los posibles números a adivinar en el juego). Por tanto, si se leyó el valor -999999, sabemos que no hemos adivinado ningún número.



SECCION: INTRODUCCION AL BASIC

TEMA: INTRODUCCION DE DATOS

Antes de volver a la línea 10 para otra búsqueda, necesitamos hacer que la nueva "READ" comience al principio de los datos. Esta función la realizamos con el "RESTORE". Después que se "encuentra un "RESTORE" los próximos datos a leer comenzarán por el primer dato de la primera instrucción DATA.

Las instrucciones DATA pueden colocarse en cualquier lugar del programa. Sólo las instrucciones READ pueden hacer uso de las instrucciones DATA en un programa, y cualquier otra que haga referencia a ella durante la ejecución es ignorada.

Una cadena (string), es una lista de caracteres. Por ejemplo, "ROCKWELL", "R 6500", "ESTO ES UNA PRUEBA", son cadenas. Al igual que las variables numéricas, las variables cadena pueden tomar valores específicos. Las variables cadena se distinguen de las numéricas por llevar un "$" a continuación del nombre de variable.

Por ejemplo:

```
A$ = "ROCKWELL R6500"
PRINT A$
ROCKWELL R6500
```

En este ejemplo, asignamos a la variable cadena A$ el valor "ROCKWELL R6500". Nótese que la cadena de caracteres que asignemos a A$, se encierra entre comillas.

FUNCION LEN

Ahora que hemos fijado A$ a un valor, podemos averiguar la longitud de este valor (el número de caracteres que contiene). Procederemos del siguiente modo:

```
PRINT LEN(A$), LEN(" MICROCOMPUTER ")
  14          13
```

La función "LEN" devuelve un entero igual al número de caracteres de la cadena.

Una cadena puede contener de 0 a 255 caracteres. Una cadena de 0 caracteres se llama cadena "nula". Antes de asignar un valor a una variable cadena, ésta está inicializada con la cadena nula. Si mandamos imprimir la cadena nula no se imprimirá ningún carácter y el cursor o el mecanismo impresor no avanzarán a la siguiente columna. Pruebe con lo siguiente:

```
PRINT LEN(Q$);Q$;3
  0  3
```

Otra manera de crear la cadena vacia: Q$ = ""
La asignación de la cadena nula a una variable puede servir para liberar el espacio ocupado por ésta cuando tenía un valor no nulo.

FUNCION LEFT$

A veces es deseable acceder a partes de una cadena y operar con ellas. Ahora que hemos fijado A$ con "ROCKWELL R 6500", podríamos querer imprimir sólo los 8 primeros caracteres de A$. Podemos hacerlo de este modo:

```
PRINT LEFT$(A$, 8)
ROCKWELL
```





"LEFT$" es una función cadena que devuelve una cadena compuesta de los N caracteres más a la izquierda de la cadena que figura como argumento. Veamos otro ejemplo:

```
FOR N = 1 TO LEN(A$) PRINT LEFT$(A$,N): NEXT N

R
RO
ROC
ROCK
ROCKW
ROCKWE
ROCKWEL
ROCKWELL
ROCKWELL
ROCKWELL R
ROCKWELL R6
ROCKWELL R65
ROCKWELL R650
ROCKWELL R6500
```

Como A$ tiene 14 caracteres, este ciclo se ejecutará para N = 1, 2, 3, . . . . . 13, 14. La primera vez sólo se imprimirá el primer carácter, la segunda vez los dos primeros, etc.

FUNCION RIGHT$

Otra función cadena llamada "RIGHT$" devuelve los N caracteres de la derecha de una expresión. Pruebe a sustituir "LEFT$" por "RIGHT$" en el ejemplo anterior y ver que sucede.

FUNCION MID$

Existe también una función que permite tomar caracteres del medio de una cadena. Probar lo siguiente:

```
FOR N=1 TO LEN(A$): PRINT MID$(A$,N): NEXT N

ROCKWELL R6500
OCKWELL R6500
CKWELL R6500
KWELL R6500
WELL R6500
ELL R6500
```

```
L  R6500
 R6500
R6500
6500
500
00
0
```

"MID$" devuelve una cadena que comienza en la enésima de A$, hasta el final (último carácter de A$).
Frecuentemente es deseable extraer sólo el enésimo carácter de una cadena. Esto se puede hacer llamando a MID$ con tres argumentos. El tercer argumento especifica el número de caracteres a devolver.

Por ejemplo:

```
FOR N= 1 TO LEN(A$):PRINT MID$(A$,N,1), MID(A$,N,2): NEXT N

R       RO
0       OC
C       CK
K       KW
W       WE
E       EL
L       LL
L       L
        R
R       R6
6       65
5       50
0       00
0       0
```

CONCATENACION DE CADENAS

Las cadenas pueden ser concatenadas (puestas una a continuación de otra) mediante el uso del operador " ". Pruebe lo siguiente:

```
B$ = "BASIC FOR"+ "  " +A$
PRINT B$
BASIC FOR ROCKWELL R6500
```

La concatenación es especialmente útil cuando se desea coger trozos de una cadena, efectuar en ellos ligeras modificaciones y volverlos a juntar de nuevo. Por ejemplo:





```
C$ = LEFT$(B$,9) + "-" + MID$(B$,11,8) + "-" + RIGHT$(B$,5)
PRINT C$
BASIC FOR - ROCKWELL - R6500
```

FUNCIONES VAL Y STR$

A veces es deseable convertir un número a su representación de cadena de caracteres, y viceversa. Las funciones "VAL" y "STR$" realizan estas funciones.
Pruebe lo siguiente:

```
STRING$ = "567.8"
PRINT VAL(STRING$)
   567,8
STRING$ = STR$(3.1415)
PRINT STRING$, LEFT$(STRING$,5)
   3.1415        3.14
```

"STR$" puede usarse para realizar formateados de entrada/salida con números. Se puede convertir un número a una cadena y después utilizar LEFT$, RIGHT$, MID$ y la concatenación para reformatear el número Tal como se desee.

STR$ también puede usarse para saber cuantas columnas tomará la impresión de un número. Por ejemplo:

```
PRINT LEN(STR$ (3.157) )
   6
```

Si en una determinada apiicación el usuario tiene que teclear una pregunta como por ejemplo CUAL ES EL VOLUMEN DE UN CILINDRO DE RADIO 5.36 METROS Y ALTURA 5.1 METROS?, se puede utilizar "VAL" para extraer los valores numéricos 5.36 y 5.1 del texto de la pregunta.

FUNCION CHR$

CHR$ es una función cadena que devuelve una cadena de caracteres que contiene el equivalente alfanumérico del argumento, de acuerdo con la tabla de conversión del apéndice E. ASC toma el primer carácter de una cadena y lo convierte a su valor decimal ASCII.
Uno de los usos más corrientes de CHR$ es enviar un carácter especial a un terminal.

ORDENACION DE CADENAS

El siguiente programa ordena una lista de datos que son cadenas de caracteres e

imprime la lista ordenada. Este programa es muy parecido al que se dió antes para ordenar una lista numérica.

```
100     DIM A$(15)
110     FOR I=1  TO 15
112     READ A$(I)
114     NEXT I
120     F = O : I = 1
130     IF A$(I) <=A$(I+1) THEN 180
140     T$ = A$(I+1)
150     A$(I+1) = A$(I)
160     A$(I) = T$
170     F = 1
180     I = I + 1
185     IF I <15  THEN 130
190     IF  F = 1  THEN 120
200     FOR I = 1  TO 15
202     PRINT A$(I)
204     NEXT I
220     DATA    AIM 65, DOG
230     DATA    CAT,  R6500
240     DATA    ROCKWELL, RANDOM
250     DATA SATURDAY, "*** ANSWER***"
260     DATA MICRO, F00
270     DATA COMPUTER, MED
280     DATA NEWPORT BEACH, DALLAS, ANAHEIM
```

OTRAS CONSIDERACIONES SOBRE CADENAS

1.– Una cadena puede contener de 0 a 255 caracteres. Todos los nombres de variables cadena acaban con un signo dólar ($), por ejemplo A$, B9$, R$, HELLO$.

2.– Las matrices de cadenas pueden ser dimensionadas exactamente igual que las matrices numéricas. Por ejemplo, DIM A$(10,10) crea una matriz de 121 elementos: once filas por once columnas (filas 0 a 10 y columnas 0 a 10). Cada elemento de la matriz es una cadena completa que puede tener hasta 255 caracteres de longitud.

| NOMBRE | EJEMPLO | PROPOSITO/USO |
|---|---|---|
| DIM | 25 DIM A$(10,10) | Asigna espacio para longitud de cada elemento de una matriz de cadenas. No reserva ningún espacio para la cadena en sí. |





| NOMBRE | EJEMPLO | PROPOSITO/USO |
|---|---|---|
| LET | 27 LET A$="F00"+U$ | Asigna el valor de una expresión de cadena a una variable cadena. LET es opcional. |
| =<br>><br><<br><= ó =<<br>>= ó =><br>< > | | Operadores de comparación de cadenas. La comparación es hecha en base a los códigos ASII, carácter a cuentra una diferencia. Si durante la comparación se alcanza el final de una de ellas, la más corta se considera la menor. Nótese que "A  " es mayor que "A" ya que los blancos de cola son significativos. |
| + | 30 LET Z$ = R$+Q$ | Concatenación de cadenas. El resultado debe tener una longitud menor que 256, de lo contrario se producirá un error LS. |
| INPUT | 40 INPUT X $ | Lee una cadena desde el teclado. La cadena no necesita estar encerrada entre comillas; en este caso se ignoran los blancos de cabecera y la cadena deberá terminar con "," ó ":". |
| READ | 50 READ X$ | Lee una cadena desde sentencias DATA contenidas en el programa. Las cadenas no necesitan ir entre comillas: en este caso se terminan con "," o con ":" y se ignoran los blancos de cabecera. Véase DATA (Tema 305) para el formato de la cadena de datos. |
| PRINT | 60 PRINT X$<br>70 PRINT "F00" +AS | Imprime la expresión de cadena en la impresora o el display. |

## SECCION: DEFINICION DE LAS INSTRUCCIONES

## TEMA: CARACTERES ESPECIALES

| CARACTER | USO |
|---|---|
| @ | Borra la línea que está siendo introducida e introduce un CR/LF (retorno de carro/avance de línea. |
| DEL | Borra el último carácter introducido. Si no va a haber más caracteres a la derecha teclear CR/LF. |
| RETURN | Cualquier línea que se introduzca debe terminar con un RETURN. Vuelve el cursor a la primera posición de la izquierda e imprime la línea si la impresora está activada. |
| F1 | Interrumpe la ejecución de un programa o de un comando de listado. F1 tiene efecto cuando termina la ejecución de una instrucción, o cuando termina la impresión de una línea en el caso de que se emplee para interrumpir un comando LIST. En ambos casos se retorna al nivel de comando del BASIC.<br><br>Se imprime "BREAK IN LINE XXXX", donde XXXX es el número de la primera línea que quedó sin ejecutar.<br><br>No existe la tecla F1 en un TTY. Sin embargo cuando se utiliza un TTY, la tecla F1 del AIM 65 está activa y se puede utilizar. |
| : | Los dos puntos se utilizan para separar instrucciones en una línea. Pueden ser utilizados en instrucciones directas e indirectas. La única limitación al número de instrucciones por línea es la longitud de ésta. No es posible hacer que un GOTO o GOSUB vayan al medio de una línea. |
| ? | Este símbolo es equivalente a PRINT. Por ejemplo,? 2+ 2 es lo mismo que PRINT 2 + 2. Puede también usarse en instrucciones indirectas. 10 ? X, al ser listada, se imprimirá 10 PRINT X. |
| $ | Este signo como sufijo de un nombre de variable establece que se trata de una variable cadena. |
| % | Como sufijo de un nombre de variable establece que se trata de un entero. |



SECCION: DEFINICION DE LAS INSTRUCCIONES

TEMA: CARACTERES ESPECIALES

| CARACTER | USO |
|---|---|
| ! | Un signo !, como sufijo de un comando INPUT, PRINT ó ?, origina que la entrada o salida se imprima aunque esté desactivada la impresora. |
| ESC | Devuelve el control al Monitor. |
| CNTL PRINT | Desactiva la impresora si estaba conectada o la activa si estaba desactivada. |

| SIMBOLO | EJEMPLO DE PRUEBA | PROPOSITO/USO |
|---|---|---|
| | A = 100 | Asigna valor a una variable. |
| | LET Z = 2,5 | LET es opcional. |
| | B = −A | Negación. Adviértase que 0−A es resta mientras que −A es negación. |
| ∧(Tecla F3) | 130 PRINT X∧3 | Exponenciación (igual a X * X * X en el ejemplo de prueba) 0 ∧ 0 = 1, cero elevado a cualquier otra potencia = 0 |
| * | 140X R* (B+D) | Multiplicación. |
| / | 150 PRINT X/1.3 | División. |
| + | 160 Z=R+T+Q | Suma. |
| − | 170 J= 100−I | Resta. |

REGLAS PARA EVALUACION DE EXPRESIONES

1) Las operaciones de mayor prioridad son realizadas antes que las de menor prioridad. Esto quiere decir que las multiplicaciones y divisiones se hacen antes que las sumas y restas. Por ejemplo, 2 + 10/5 es igual a cuatro, no a 2.4. Cuando en una fórmula se encuentran operaciones de la misma prioridad, se ejecuta primero la situada más a la izquierda: 6 − 3 + 5 = 8 y no −2.

2) El orden de ejecución de las instrucciones puede especificarse explícitamente, utilizando paréntesis. Por ejemplo, para sumar 5 y 3 y dividir el resultado por 4, podríamos poner (5 + 3) /4, que es igual a 2. En cambio, si hubiésemos puesto 5 + 3/4 habríamos obtenido 5.75 como resultado (5 + 3/4).





La prioridad de los operadores utilizados en la evaluación de expresiones, es como sigue, comenzando con los de prioridad mayor:

NOTA

Los operadores listados en la misma línea tienen la misma prioridad.

| | | | |
|---|---|---|---|
| 1) | Las expresiones entre paréntesis son siempre evaluadas las primeras. | | |
| 2) | ∧ (Tecla F3) | Exponenciación. | |
| 3) | Negación | −X, donde X puede ser una fórmula. | |
| 4) | * y / | Multiplicación y división. | |
| 5) | + y − | Suma y resta. | |
| 6) | OPERADORES DE RELACION: (igual prioridad para todos) | = | igual |
| | | <> | distinto |
| | | < | menor que |
| | | > | mayor que |
| | | =< ó <= | menor que ó igual |
| | | => ó >= | mayor que ó igual |

(Los tres siguientes son operadores lógicos).

| | | |
|---|---|---|
| 7) | NOT | El Operador lógico, orientado a bits, "NOT" al igual que la negación únicamente toma la fórmula a su derecha como argumento. |
| 8) | AND | Operador lógico, orientado a bits, "AND". |
| 9) | OR | Operador lógico, orientado a bits, "OR". |

Una expresión relacional se puede usar como parte de cualquier expresión.

Las expresiones de operadores relacionales siempre tomarán un valor de Verdadero (−1) o Falso (0). Por ejemplo, (5 = 4) = 0, (5 = 5) = −1, (4 > 5) = 0, (4 < 5) = −1.

La cláusula THEN de una instrucción IF, es ejecutada cuando la fórmula que sigue a IF no es igual a 0 . Esto es: IF X THEN . . . es equivalente a IF X<> 0 THEN . . .

| SIMBOLO | EJEMPLO | PROPOSITO/USO |
|---|---|---|
| = | 10 IF A=15 THEN 40 | Expresión igual a expresión |
| < > | 70 IF A<>0 THEN 5 | Expresión distinta de expresión |
| > | 30 IF B > 100 THEN 8 | Expresión mayor que expresión |
| < | 160 IF B < 2 THEN 10 | Expresión menor que expresión |
| <=, =< | 180 IF 100<= B C THEN 10 | Expresión menor que/o igual a expresión |
| >=, => | 190 IFQ =>R THEN 50 | Expresión mayor que o igual a expresión |
| AND | 2 IF A <5 AND B < 2 THEN 7 | Si expresión 1 (A < 5) y expresión 2 (B < 2) son ambos ciertos, bifurcar a la línea 7 |
| OR | IF A <1 OR B<2 THEN 2 | Si alguna de las expresiones (A < 1) ó (B < 2) es cierta, bifurcar a la línea 2 |
| NOT | IF NOTQ3 THEN 4 | Si la expresión "NOTQ3" es cierta (porque Q3 es falsa), ir a la línea 4. NOTA: NOT –1= 0 (No cierta: falso). |

AND, OR y NOT pueden utilizarse para manipular bits y para realizar operaciones booleanas.

Estos tres operadores convierten sus argumentos a 16 bits, en forma de entero con signo en formato de complemento a 2, en el rango de –32768 y + 32767. A continuación realizan la operación lógica especificada, y devuelven el resultado dentro del mismo rango. Si los argumentos no están dentro del rango, resultará un error "FC".





Las operaciones son realizadas en el modo bit a bit, esto es que cada bit del resultado se obtiene examinando el bit de la misma posición para cada argumento.

La siguiente tabla de verdad, muestra las relaciones lógicas entre bits.

| OPERADOR | ARGUMENTO 1 | ARGUMENTO 2 | RESULTADO |
|---|---|---|---|
| AND | 1 | 1 | 1 |
| | 0 | 1 | 0 |
| | 1 | 0 | 0 |
| | 0 | 0 | 0 |
| OR | 1 | 1 | 1 |
| | 1 | 0 | 1 |
| | 0 | 1 | 1 |
| | 0 | 0 | 0 |
| NOT | 1 | - | 0 |
| | 0 | - | 1 |

EJEMPLOS: (En todos estos ejemplos, no aparecen los ceros de cabecera de los números binarios).

63 AND 16 = 16: Puesto que 63 es igual en binario a 111111 y 16 a 10000, el resultado de un AND es el binario 10000, esto es 16.

15 AND 14 = 14 15 es igual a 1111 en binario y 14 es igual a 1110 luego AND 14 es igual a 1110 o sea 14.

–1 AND 8 = 8 –1 en binario es 1111111111111111 y 8 es 1000, por consiguiente el resultado en binario es 1000, esto es 8 decimal.

4 AND 2 = 0 4 es 100 en binario y 2 es 10, luego el resultado es 0 ya que ninguno de los bits del mismo orden de los argumentos coinciden en ser 1 simultáneamente, para que el resultado diera 1.

4 OR 2 = 6 El binario 100 operado con "OR" con el binario 110, resulta el binario 110, ó 6 decimal.

| | |
|---|---|
| 10 OR 10 = 10 | El binario 1010 operado con OR, con el binario 1010 resulta 1010, ó 10 decimal. |
| −1 OR −2 = −1 | El binario 1111111111111111 (−1) operado mediante OR con el binario 1111111111111110 (−2) da el binario 1111111111111111 ó −1. |
| NOT 0 = −1 | El complemento bit a bit de 16 ceros binarios es 16 unos esto es 1111111111111111, ó bien (−1). También NOT−1 = 0. |
| NOT X | NOT X es igual a −(X 1). Esto es así porque para formar el complemento a 2 de un binario de 16 bits se complementa cada bit y se suma 1. |
| NOT 1 = −2 | El complemento en 16 bits de 1 es 1111111111111110, que es igual a −(1 + 1) o sea −2. |

Una utilización típica de las operaciones bit a bit es el chequeo de los bits de una determinada posición del micro que refleja el estado de aglún dispositivo externo.

La posición 7 de un byte es la del bit más significativo mientras que la posición 0 corresponde al menos significativo.

Por ejemplo, supongamos que el bit 1 de la posición 40963 es 0 cuando la ventana de la habitación x está cerrada, y 1 si está abierta. El siguiente programa imprimirá "ALERTA INTRUSOS" si se abre la ventana:

```
10 IF NOT (PEEK(40963) AND 2) THEN 10
```

Esta línea se ejecutará repetidamente hasta que el bit 1 (enmascarado o seleccionado por el 2) valga 1. Cuando ocurra esto, vamos a la línea 20.

```
20 PRINT "ALERTA, INTRUSOS"
```

La línea 20 sacará este mensaje.

Sin embargo, podemos sustituir la sentencia 10, por una instrucción "WAIT" que tiene exactamente el mismo efecto.

```
10 WAIT 40963, 2
```

Esta línea retrasa la ejecución de la siguiente instrucción del programa hasta que el bit 1 de la posición A003 valga 1. El WAIT es mucho más rápido que la sentencia IF equivalente y además ocupa menos bytes de almacenamiento de programa.





Lo siguiente es otra manera interesante de usar los operadores relacionales:

```
125 A = −(B >C) *B −  (B <=C) *C
```

Esta instrucción asignará a la variable A el MAX (B,C) = a la mayor de las variables B y C.

## SECCION: DEFINICIONES DE SENTENCIAS

## TEMA: COMANDOS

Se puede introducir un comando BASIC cuando aparece el cursor. A esto se le llama el "Nivel de Comando". Los comandos se pueden utilizar como instrucciones de programa. Algunos comandos como LIST, NEW y LOAD al terminar de realizarse originan que el programa acabe su ejecución. Cada comando puede requerir uno o más argumentos además del propio comando tal como se refleja en la columna Sintaxis/Función.

Un argumento sin paréntesis precisa que sea introducido precisamente así. Igualmente los argimentos que figuran entre paréntesis deben introducirse con paréntesis. Los argumentos entre corchetes, son opcionales y si se incluyen, deben introducirse con o sin paréntesis, según aparezcan a continuación:

| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| CLEAR | CLEAR<br><br>Limpia todas las variables del programa, reinicia los estados de las "FOR" y "GOSUB" y restaura los "DATA" | CLEAR |
| CONT | CONT<br><br>Continúa la ejecución del programa después de haber tecleado F1 o de que una instrucción INPUT o STOP haya finalizado la ejecución. No se puede continuar después de un error, después de una modificación del programa o antes de que el programa haya sido ejecutado. Uno de los principales propósitos de CONT es la depuración. Supongamos que se inicia la ejecución de un programa y pasado un tiempo, todavía no se ha impreso nada. Esto puede deberse a que el programa está haciendo un cálculo largo, pero también puede ocurrir que haya entrado en un "ciclo infinito".<br><br>Un "ciclo infinito" es una serie de instrucciones BASIC que no tienen salida. El BASIC estará ejecutando una y otra vez estas instrucciones continuamente, hasta que usted intervenga o hasta que se desconecte el AIM 65. Si sospecha que un programa está en un ciclo infinito, pulse F1 hasta que aparezca el mensaje BREAK. Aparecerá también el número de Línea que se estaba ejecutando. Cuando el BASIC haya presentado el cursor se puede utilizar PRINT para ver qué valores han tomado algunas de las variables. Después de este examen, quizás esté satisfecho de como marcha su programa.<br><br>Entonces puede teclear CONT para que se continúe ejecutando donde se quedó, o bien dar GOTO para que reinicie la ejecución en otra línea. También podrá utilizar sentencias de asignación para dar valores diferentes a sus variables.<br><br>Recuerde que si interrumpe un programa con la tecla F1 y espera continuarlo más adelante, no debe cometer errores ni introducir nuevas líneas en el programa. Si lo hace, no le será posible continuar y obtendrá un error "CN" (continue not). Es imposible continuar la ejecución de un comando directo. CONT siempre reanuda la ejecución en la primera instrucción sin ejecutar cuando se interrumpió el programa con F1. | |





| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| FRE | FRE (expresión)<br><br>Dá el número de bytes de memoria que no está siendo utilizado actualmente por el BASIC. Se debe utilizar un argumento ficticio (0 ó 1). | 270 PRINT FRE(0) |
| LIST | LIST [(desde)] [-(hasta)]<br><br>Listar el programa, empezando opcionalmente en una línea especificada. LIST | |

| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| | puede ser interrumpido por la tecla F1 (El BASIC terminará de imprimir la línea en curso). | |
| | Listar el programa completo | LIST |
| | Listar sólo la línea 100 | LIST 100 |
| | Listar desde la línea 100 a 1000 | LIST 100–1000 |
| | Listar desde la línea en curso hasta la 1000 | LIST–1000 |
| | Listar desde la línea 100 al fin del programa | LIST 100– |
| LOAD | LOAD<br>Cargar un programa desde la cinta de casette. Cuando se ha terminado, LOAD hará que aparezca el cursor. Ver apéndice G para más información. | LOAD |
| NEW | NEW<br>Borra el programa en curso y todas las variables | NEW |
| PEEK | PEEK (dirección)<br><br>La función PEEK devuelve el contenido de la dirección de memoria I en decimal. El valor devuelto será => 0 y <= 255. Si I es > 65535 o < 0, se producirá un error FC. Si intenta leer de una dirección de memoria no existente, devolverá un valor imprevisible. | 356 PRINT PEEK(I) |
| POKE | POKE posición, byte<br><br>La instrucción POKE, almacena el byte especificado como segundo argumento, en la posición que se indica como primer argumento. El byte que se va a almacenar debe ser=>0 y<= 255, de lo contrario se producirá error FC.<br>Precaución: Si no se usa con cuidado, la instrucción POKE, puede ocurrir que el | 357 POKE I,J |





| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| | programa, el BASIC o las funciones del Monitor, operen incorrectamente, que se queden colgados o bien que se pierda el programa.<br><br>Recuerde que las páginas 0 y 1 de memoria están reservadas para ser usadas por el BASIC, y por tanto no deberían utilizarse para almacenar variables del programa. Un POKE a una posición de memoria que no exista, no producirá ningún perjuicio. Uno de los principales usos de POKE es pasar argumentos a subrutinas de lenguaje máquina. (Ver apéndice F). También podrán utilizarse las instrucciones PEEK y POKE para escribir un diagnóstico de memoria o un ensamblador en BASIC. | |
| RUN | RUN número de línea<br>Comienza la ejecución del programa que está en memoria, desde la línea especificada. RUN borra todas las variables (hace un CLEAR) y restaura los DATA. Si ha parado el programa y desea reanudar la ejecución en cierto punto, utilice un GOTO directo para que comience en la línea deseada, o CONT para continuar desde la ruptura.<br><br>Comienza la ejecución del programa desde la línea de número menor. | RUN 200 |
| SAVE | SAVE<br>Guarda el programa en la memoria del AIM 65, en la cinta del cassette. El programa se deja allí sin ningún cambio. Se puede guardar más de un programa, utilizando este comando. Ver apéndice G para más información. | SAVE |

En la descripción de instrucciones siguiente, los argumentos B, C, V ó W denotan una variable numérica, X denota una expresión numérica, X$ denota una expresión de cadena e I y J denotan una expresión que es truncada a entero antes de que se ejecute la instrucción. El truncamiento significa que la parte fraccionaria del número se pierde. P.e. 3.9 se convierte en 3, 4 01 a 4 etc.

Una expresión es una serie de variables, operadores, llamadas a funciones y constantes que, después de que se han realizado las operaciones y las llamadas a funciones, utilizando las reglas de prioridad, toma un valor numérico o de cadena a caracteres.

Una constante es un número (3.14) o una cadena de caracteres ("FOO").

| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| DEF | DEF FNx ((lista de argumentos)) expresión<br>El usuario puede definir funciones como las internas (SQR, SGN, ABS, etc.) mediante el uso de la instrucción DEF. | 100 DEF. FNA(V) = V/B + C |
| | El nombre de la función es FN, seguido de un nombre de variable legal, por ejemplo "FNX, FNI7, FNKO, FNR2. | |
| | Las funciones definidas por el usuario están restringidas a una línea. Una función puede definirse mediante cualquier expresión, pero sólo puede tener un argumento. En el ejemplo B y C son variables que se usan en el programa. La ejecución de la instrucción DEF, define la función. Las funciones de usuario pueden ser redefinidas ejecutando otro DEF para la misma función. "V" es llamada "Variable ficticia". | |
| | La ejecución de esta instrucción, a continuación de la anterior dará lugar que Z valga 3/B + C, sin embargo el valor de V no cambiará. | 105 Z = FNA (3) |
| DIM | DIM variable (tamaño 1, tamaño 2 . . .). Reserva espacio para matrices. (Una | 113 DIM A(3), B(10) |





| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| | sentencia DIM pone a cero todos los elementos de la matriz). | |
| | Las matrices pueden tener de 1 a 255 dimensiones. | 114 DIM R3(5,5) D$(2,2,2) |
| | Las matrices se pueden dimensionar dinámicamente durante la ejecución del programa. Si una matriz no está explícitamente dimensionada con un DIM, se asume que es una matriz de una dimensión, cuyo único índice va de 0 a 10 (11 elementos). | |
| | Si aparece esta instrucción antes de una instrucción DIM para A, sería como si previamente a la línea 117 se hubiera ejecutado un DIM A (10). Todos los subíndices comienzan con cero (0), lo cual quiere decir que DIM X (100) realmente crea una matriz de 101 elementos. | 117 A(8)=4 |
| END | END | 999 END |
| | Termina la ejecución de un programa sin que se imprima un mensaje BREAK (Ver STOP). CONT, después de una instrucción END hace que se reanude la ejecución en la siguiente instrucción a END. END se puede usar en cualquier lugar del programa, y es opcional. | |
| FOR | FOR variable expresión TO expresión STEP expresión (Ver sentencia NEXT) | 300 FOR V = 1 TO 9.3 STEP.6 |
| | Se asigna a V el valor de la expresión que sigue al signo igual, en este caso 1. Este valor se llama valor inicial. A continuación se ejecutan las instrucciones entre FOR y NEXT. El valor final es el valor de la expresión que sigue a TO. El paso o incremento es el valor de la | |

| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| | expresión que sigue a STEP. Cuando se encuentra una instrucción NEXT, se añade el paso a la variable. | |
| | Si no se especifica STEP, se asume que vale 1. Si el paso es positivo y el nuevo valor de la variable es < = que el valor final (9, 3, en el ejemplo), o el valor del paso es negativo y el nuevo valor de la variables es = > que el valor final, se ejecuta la primera instrucción que sigue a NEXT. | 310 FOR V=1 TO 9.3 |
| | En todos los ciclos FOR, las instrucciones entre FOR y NEXT se ejecutan al menos una vez, incluso en casos como FOR V = 1 TO 0. | |
| | Nótese que se pueden utilizar expresiones (fórmulas), para los valores inicial, final y del step, de un ciclo FOR. El valor de estas expresiones se calculan una sola vez, antes de que se ejecute el cuerpo del ciclo FOR . . . NEXT. | 315 FOR V = 10*N TO 3.4/Q STEP SQR (R) |
| | Cuando se ejcuta la instrucción que sigue a NEXT, la variable del ciclo no es nunca igual al valor final, sino que es igual al valor que haya originado la terminación del ciclo. Las instrucciones entre FOR y su correspondiente NEXT en los dos ejemplos anteriores (310 y 320) serían ejecutados nueve veces. | 320 FOR V=9 |
| | Error: no utilizar ciclos FOR . . . NEXT anidados con la misma variable índice. | 330 FOR W=1 TO10: FOR W=1 TO5; NEXT W: NEXT W |
| | El anidamiento de ciclos FOR, está únicamente limitado por la memoria disponible (Ver apéndice C.). | |
| GOSUB | GOSUB número de línea | 10 GOSUB 910 |





| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| | Bifurca a la instrucción especificada (910) hasta que encuentre un RETURN; entonces se vuelve a bifurcar a la instrucción que sigue a GOSUB. El anidamiento de GOSUB está limitado únicamente por la memoria disponible. | |
| GOTO | GOTO número de línea<br>Salta a la instrucción especificada. | 50 GOTO 100 |
| IF . . . GOTO | IF expresión GOTO número de línea . . . | 32 IF X < =Y + 23.4 GOTO 92 |
| | Equivalente a IF . . . THEN, excepto que IF . . . GOTO debe ser seguido por un número de línea, mientras que IF . . . THEN puede estar seguido de un número de línea o de otra instrucción. | |
| IF . . . THEM | IF expresión THEN número de línea. Bifurca a la instrucción especificada si la relación es cierta. | IF X < 210 THEN 5 |
| | Ejecuta todas las instrucciones que siguen al THEN si la relación es cierta. | 20 IFX < 0 THEN PRINT "X MENOR QUE 0" |
| | AVISO: "Z = A" no se ejecutará nunca porque si la relación es cierta, BASIC bifurcará a la línea 50. Si la relación es falsa, BASIC continuará con la línea que sigue a la 25. | 25 IFX=5 THEN 50: Z=A |
| | En este ejemplo, si x es menor que 0, se ejecutará la instrucción PRINT y después el GOTO hará que se bifurque a la línea 350. Si x era 0 o positivio, el BASIC continuará con la línea que siga a la 26. | 26 IF X<0 THEN PRINT "ERROR X NEGATIVO": GOTO 350 |
| LET | LET variable = expresión. Asigna un valor a una variable. "LET" es opcional. | 300 LET W=X<br>310 V=5.1 |
| NEXT | NEXT [variable] [, variable] . . .<br>Marca el final de un ciclo FOR. | 340 NEXT V |

| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| | Si no se especifica una variable, se toma la del ciclo FOR más reciente. | 345 NEXT |
| | Puede utilizarse un único NEXT para varias sentencias FOR. Equivalente a NEXT V: NEXT W. | 350 NEXT V, W |
| ON . . . GOSUB | ON expresión GOSUB línea [, línea] . . | 110 ON I GOSUB 50, 60 |
| | Idéntico a ON . . . GOTO excepto porque se ejecuta una llamada a subrutina en lugar de un GOTO. El RETURN de la subrutina retorna a la instrucción que sigue a ON . . . GOSUB. | |
| ON . . . GOTO | ON expresión GOTO línea[, línea] . . Bifurca a la línea especificada por el i–ésimo número detrás del GOTO. Esto es<br>IFI = 1 THEN GOTO 10<br>IFI = 2 THEN GOTO 20<br>IFI = 3 THEN GOTO 30<br>IFI = 4 THEN GOTO 40 | 100 ON I GOTO 10, 20,30,40 |
| | Si I = 0 o I intenta seleccionar una línea que no existe (I > = 5, en este caso), se ejecuta la instrucción que sigue al ON. | |
| | Por otra parte si I es > 255 ó < 0, se producirá un error FC. En un ON . . . GOTO pueden ir tantos números de línea como quepan en una línea. | |
| | Esta instrucción bifurcará a la línea 40 si la expresión x es menor que cero, a la línea 50 si es igual a cero, y a la 60 si es mayor que cero. | 105 ON SGN (X) + 2 GOTO 40, 50, 60. |
| REM | REM cualquier texto<br>Permite al programador poner comentarios a sus programas. Las instrucciones | 500 REM HAGO QUE V = 0 |





| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| | REM no se ejecutan, pero puede bifurcarse a ellas. Una instrucción REM se termina con un fin de línea, pero no por un ":". | |
| | En este caso V = 0 no se ejecutará nunca. | 505 REM PONER V = 0; V = 0 |
| | En este caso se ejecutará V = 0. | 505 V = 0 REM PONER V = 0 |
| RESTORE | RESTORE | 510 RESTORE |
| | Permite la relectura de las instrucciones DATA. Después de un RESTORE, el siguiente item de datos a leer será el primero de la primera instrucción DATA que haya en el programa. El segundo será el que aparezca en segundo lugar en la primera DATA y así sucesivamente de acuerdo con el funcionamiento de la operación READ. | |
| RETURN | RETURN<br>Hace que una subrutina retorne a la instrucción siguiente al último GOSUB ejecutado. | 50 RETURN |
| STOP | STOP<br>Hace que un programa pare su ejecución y que entre el modo de comando.<br>Imprime BREAK IN LINE 900.<br>(Para el caso del ejemplo). Un CONT después de STOP conduce a la instrucción que sigue a STOP. | 900 STOP |
| USR | USR (argumento).<br>Llama a la subrutina del usuario en lenguaje de ensamblador con un argumento. Ver PEEK y POKE en el tema 303, y el apéndice F. | 200 V = USR(W) |

| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| WAIT | WAIT dirección, máscara [,argumento de selección]<br>Esta instrucción lee el contenido de la posición direccionada hace un OR exclusivo con el argumento de selección, y a continuación un AND del resultado con la máscara. Esta secuencia se repite hasta que se obtiene un resultado distinto de cero, momento en el cual la ejecución continúa con la instrucción siguiente a WAIT. Si el WAIT no tiene argumento de selección, se asume que su valor es cero. Si usted espera que un bit sea cero el argumento de selección debe tener un "uno" en el bit correspondiente. El valor de selección (K) y el de la máscara (J) pueden valer de 0 a 255. La dirección (I) puede valer de 0 a 65535. | 805 WAIT I,J,K<br>806 WAIT I,J |





| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| DATA | DATA item [,item . . . .]<br>Especifica datos leidos de izquierda a derecha. La información aparece en las instrucciones DATA en el mismo orden en que será leída en el programa. | 10 DATA 1,3, –1E3, .04 |
| | Se pueden leer cadenas de caracteres desde instrucciones DATA. Si quiere que la cadena contenga blancos en cabeza, el carácter (:), o comas (,), se debe encerrar la cadena entre comillas ("). Es ilegal que haya comillas dentro de una cadena (" "BASIC" " es ilegal). | 20 DATA "F00", E1 |
| INPUT | INPUT [!] ["cadena de caracteres de advertencia";], variable [,variable].<br>Solicita que se introduzcan datos desde el teclado. Cada valor debe estar separado del que le sigue por una coma (,). El último valor introducido debe ir seguido de un retorno de carro. Se visualiza un "?" como carácter de advertencia. Sólo se deben de introducir constantes como respuesta a una instrucción INPUT, como 4. 5 E–3 o "CAT". Si una sentencia INPUT necesita más datos que los que se han introducido, aparecerá un "??" y entonces se le darán los datos que faltan. Si se dan más datos de los precisos, aparecerá el mensaje "EXTRA IGNORED". Las cadenas deben entrar en el mismo formato en que son especificados en las instrucciones DATA. | 3 INPUT V, W, W2 |
| | Opcionalmente presenta una cadena de caracteres de advertencia ("VALOR") antes de que solicite datos de entrada desde el teclado. Si se pulsa RETURN como respuesta a una instrucción INPUT, el BASIC vuelve al modo de comando. Tecleando CONT después de | 5 INPUT "VALOR"; V |

| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| | haber interrumpido un INPUT, se reanudará la ejecución en la instrucción INPUT.<br><br>Si se incluye el carácter ! a continuación de INPUT, se imprimirán tanto la cadena de caracteres de aviso, como las entradas del usuario aunque la impresora esté desactivada. | |
| PRINT | PRINT [ ! ] expresión [ ,expresión ] imprime el valor de las expresiones en el display/impresora. Si la lista de valores a imprimir no terminan con una coma (,) ó punto y coma (;) se ejecutará un retorno de carro/avance de línea, después de que se hayan impreso todos los valores. También se pueden imprimir cadenas encerradas entre comillas. Si dos expresiones están separadas por punto y coma (;) sus valores se imprimen uno a continuación de otro. Si después de una expresión aparece una coma y la cabeza de impresión está en la posición 11 o superior, se ejecuta un retorno de carro y avance de línea.<br>Si la cabeza de impresión está antes de la posición 11, se imprimen espacios hasta que el carro se sitúa al comienzo del siguiente campo de 10 columnas. Si hay una cadena de blancos entre comillas, como en la línea 370 de los ejemplos, se ejecutará un retorno de carro/ avance de línea. | 260 PRINT X, Y; Z<br>370 PRINT " "<br>380 PRINT X, Y;<br>390 PRINT "EL VALOR ES"; A<br>400 PRINT A2, B |
| | "EL VALOR ES" será visualizado e impreso. | 410 PRINT "EL VALOR ES"; A |
| | Se pueden imprimir expresiones de cadenas | 420 PRINT MID$ (A$, 2); |
| READ | READ variable [ ,variable ] | 490 READ V, W |





| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| | Lee datos de una instrucción DATA, en las variables especificadas. El primer item de datos leídos será el primero que aparezca en la primera lista DATA del programa. El segundo será el segundo item de la primera DATA y así sucesivamente. Cuando se han leído todos los datos de la primera DATA, el siguiente item de datos que se leerá será el primero que aparezca en la segunda DATA del programa. Si se intenta leer más datos de los que existen en instrucciones DATA de un programa se producirá un error OD (out of data). | |
| SPC | SPC (expresión)<br>Imprime I espacios (o blancos) en el terminal. Unicamente debe ser usado en una instrucción PRINT. I debe ser >= 0 y < = 255 de lo contrario se producirá un error FC. | 250 PRINT SPC(I) |
| TAB | TAB (expresión)<br>Produce un espaciado a la posición o columna de impresión especificada. Sólo se puede usar con PRINT. La posición de impresión más a la izquierda es 0 y la más a la derecha 19. Si el carro está más adelante de la posición I, no se imprime nada. I debe ser >=0 y <=255.<br>Si I es mayor que 19, la impresora saltará el número de líneas precisas para alcanzar la posición especificada. | 240 PRINT TAB(I) |

| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| ASC | ASC (cadena de caracteres).<br>Devuelve el valor numérico del primer carácter de la cadena X$. Ver apéndice E para la tabla de conversión ASCII/números. Se produce un error FC si X$ es la cadena nula. | 300 PRINT ASC(X$) |
| CHR$ | CHR$ (expresión)<br>Devuelve un carácter, el equivalente ASCII del argumento (I), que debe ser un número entre 0 y 255. Ver apéndice E. | 275 PRINT CHR$(I) |
| GET | GET variable de cadena de caracteres.<br>Introduce un carácter desde el teclado. Si el dato, está disponible en el teclado, se asigna este a la variable indicada en la instrucción GET. Si no hay dato disponible, continuará la ejecución del programa BASIC.<br>GET sólo puede usarse como un comando indirecto. | 10 GET A$ |
| LEFT$ | LEFT$ (expresión de cadena, longitud)<br>Entrega los I primeros caracteres de la izquierda de la cadena X$. Si I<= 0 o < 255 se producirá un error FC. | 310 PRINT LEFT$ (X$,I) |
| LEN | LEN (cadena de caracteres).<br>Entrega la longitud de la cadena X$ en caracteres (bytes), contando tanto los espacios como los caracteres no imprimibles. | 330 PRINT LEN (X$) |
| MID$ | MID$ (cadena de caracteres, principio, [longitud])<br>Si MID$ es llamada con dos argumentos, esta devolverá los caracteres de la expresión de la cadena X$ desde la posición I a la derecha; si I > LEN (X$), entonces MID$ devolverá una cadena vacia. Si I <=0 o> 255 se producirá un error FC. | 330 PRINT MID$ (X$, I) |





| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| | Si se llama a MID$ con tres argumentos, esta devolverá una cadena de J caracteres de X$ empezando en el carácter nº. I. Si I > LEN (X$), entonces MID$ devolverá una cadena vacia. Si I o J <=0 o 255 se producirá un error FC.<br>Si J indica más caracteres de los que haya a la derecha del carácter nº I se devolverán todos los caracteres desde el Nº. I en adelante. | 340 PRINT MID$ (X$. I, J) |
| RIGHT$ | RIGHT$ (cadena de caracteres, longitud)<br>devuelve los I caracteres más a la derecha de la cadena X$. Cuando I<=0 o >255 ocurrirá un error FC. Si I >= LEN (X$) entonces RIGHT$ devuelve toda la cadena X$. | 320 PRINT RIGHT$ (X$, I) |
| STR$ | STR$ (expresión)<br>Devuelve una cadena que es la representación en caracteres, de la expresión numérica X. Por ejemplo, STR$ (3.1) = "3.1". | 290 PRINT STR$ (X) |
| VAL | VAL (cadena de caracteres)<br>Devuelve la cadena X$ convertida en un número. Por ejemplo, VAl ("3.1.") =3.1. Si el primer carácter de la cadena, sin considerar los espacios, no es un signo más ( + ) o menos (−); un dígito o un punto decimal (.) VAL devolverá (0). | 280 PRINT VAL (X$) |

| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| ABS | ABS (expresión)<br>Entrega el valor absoluto de la expresión X. ABS entrega X si X > = 0, y – X en caso contrario. | 120 PRINT ABS(X) |
| ATN | ATN (expresión)<br>Entrega el arco tangente de la expresión X. El resultado se dá en radianes con valores comprendidos entre –PI/2 y PI/2 (PI = 3.1416).<br><br>Para utilizar esta función, debe almacenarse su programa en memoria. Ver en Apéndice H la forma de realización. | 210 PRINT ATN(X) |
| COS | COS (expresión)<br>Entrega el coseno de la expresión X. Supone que X está en radianes. | 200 PRINT COS(X) |
| EXP | EXP (expresión)<br>Entrega la constante "E" (2,71828) elevada a la potencia X (E∧X). Para no desbordar la capacidad del BASIC AIM, el máximo argumento utilizable en EXP es X = 88.0296. | 150 PRINT EXP(X) |
| INT | INT (expresión)<br>Entrega el máximo número entero igual o menor que la expresión X. Por ejemplo: INT (.23) = 0, INT (7) = 7, INT (–1) = –1, INT (1.1) = 1, INT (–1.1) = –2.<br><br>La siguiente expresión redondea X a D cifras decimales:<br>INT (X*10∧D + . 5) / 10∧D | 140 PRINT INT(X) |
| LOG | LOG (expresión)<br>Entrega el logaritmo neperiano (en base E) de la expresión X. Para obtener el logaritmo en base Y de X usar la fórmula LOG (X) /LOG (Y). Ejemplo, el logaritmo en base 10 del número 7 es LOG (7) /LOG (10). | 160 PRINT LOG(X) |





| INSTRUCCION | SINTAXIS/FUNCION | EJEMPLO |
|---|---|---|
| RND | RND (parámetro)<br>Genera un número aleatorio entre 0 y 1. El parámetro X controla la generación de números aleatorios como sigue:<br>X< 0 inicia una nueva serie de números aleatorios usando X. Llamando a RND con el mismo X inicia la misma serie de números aleatorios. X = 0 da el último número aleatorio generado. Nuevas llamadas a RND (0) devolverán siempre el mismo número aleatorio. X= 0 genera un nuevo número aleatorio entre 0 y 1. Observar que (B–A) *RND(1)+A genera un número aleatorio comprendido entre A y B. | 170 PRINT RND(X) |
| SGN | SGN (expresión)<br>Da 1 si X > 0, 0 si X = 0, y –1 si X < 0. | 230 PRINT SGN(X) |
| SIN | SIN (expresión)<br>Calcula la función trigonométrica seno de la expresión X, interpretada en radianes.<br>NOTA: COS (X) = SIN (X+3.14159/2) y que 1 Radian = 180/PI grados = 57.2958 grados, o sea que el seno de X grados = SIN (X/57.2958). | 190 PRINT SIN(X) |
| SQR | SQR (expresión)<br>Calcula la raíz cuadrada de la expresión X. Si X es negativo se producirá un error FC. | 180 PRINT SQR(X) |
| TAN | TAN (expresión)<br>Calcula la tangente de la expresión X, interpretada en radianes. | 200 PRINT TAN(X) |

FUNCIONES DERIVADAS

Las siguientes funciones, que no son intrínsecas del BASIC, pueden calcularse utilizando las funciones BASIC existentes.

## SECCION: DEFINICION DE LAS INSTRUCCIONES

## TEMA: FUNCIONES ARITMETICAS

| FUNCION | EXPRESION DE LA FUNCION EN BASIC |
|---|---|
| SECANTE | SEX (X) = 1/COS(X) |
| COSECANTE | CSC(X) = 1/SIN(X) |
| COTANGENTE | COT(X) = 1/TAN(X) |
| ARCO SENO* | ARCSIN(X) = ATN(X/SQR(−X*X+1)) |
| ARCO COSENO* | ARCCOS(X) = −ATN(X/SQR(−X*X+ 1) ) + 1.5708 |
| ARCO SECANTE* | ARCSEC(X) = ATN(SQR(X*X−1) + (SGN (X) −1)*1.5708 |
| ARCO COSECANTE* | ARCCSC(X) = ATN(1/SQR(X*X−1) + (SGN(X) |
| ARCO COTANGENTE* | ARCCOT(X) = −ATN(X)+1.5708 |
| SENO HIPERBOLICO | SINH(X) = (EXP(X) − EXP(−X))/2 |
| COSENO HIPERB. | COSH(X) = (EXP(X)+EXP(−X))/2 −1)*1.5708 |
| SECANTE HIPERB. | SECH(X) = 2/(EXP(X)+EXP(−X)) |
| COSECANTE HIPERB. | COSH(X) = 2/(EXP(X)−EXP(−X)) |
| COTANG. HIPERB. | COTH(X) = EXP(−X)/(EXP(X)−EXP(−X))* 2+1 |
| ARG. SENO HIP. | ARGSINH(X) = LOG (X + SQR(X*X + 1)) |
| ARG. COSENO HIP. | ARGCOSH(X) = LOG (X + SQR(X*X−1)) |
| ARG. TANGENTE HIP. | ARGTANH(X) = LOG ((1 + X)/(1−X) /2 |
| ARG. SECANTE HIP. | ARGSECH(X) = LOG ((SQR(−X*X+1)+1)/X) |
| ARG. COSECANTE HIP. | ARGCSCH(X) = LOG((SGN(X)*SQR(X*X 1) + 1)/X) |
| ARG. COTANGENTE HIP. | ARGCOTH(X) = LOG ((X+1)/(X−1))/2 |

* Estas funciones necesitan la función ATN definida por el usuario. Ver Apéndice H.



## SECCION: APENDICES

## TEMA: MENSAJES DE ERROR

Si se produce un error, el Basic dará un mensaje de error, volverá al nivel de comandos y presentará el cursor. Los valores de las variables y el texto del programa permanecerán intactos, pero el programa no puede continuar y se pierde el contexto de todas las GOSUB y FOR.

Cuando ocurre un error en una instrucción directa, no se imprime el número de línea.

Formatos de mensaje de error:

| | | | |
|---|---|---|---|
| Instrucciones directas | ? | XX | ERROR |
| Instrucciones indirectas | ? | XX | ERROR IN YYYY |

En los dos ejemplos anteriores, "XX" indica el código de error. En una instrucción indirecta "YYYY" indica el número de la línea donde se dió el error.

La tabla siguiente recoge los posibles códigos de error y su significado:

| CODIGO DE ERROR | SIGNIFICADO |
|---|---|
| BS | Subíndice inadecuado. Se intenta referenciar un elemento de una matriz fuera de la dimensión de la matriz. El error puede ocurrir si se utilizan una cantidad equivocada de dimensiones al referirse a una matriz; por ejemplo, LET A(1,1,1) = Z cuando se dimensionó DIM A (2,2). |
| CN | Error continuado. Se intentó continuar un programa inexistente, o después de ocurrido un error, o después de incluir una nueva línea en el programa. |
| DD | Doble Dimensionado. Después de dimensionada una matriz, se encontró otra instrucción DIM para la misma matriz. Este error ocurre frecuentemente cuando una matriz es dimensionada automáticamente a 10, por no estar pre dimensionada al encontrar una instrucción como A(I) =3, y posteriormente el programa incluye una DIM A (100). |
| FC | Error de llamada de función. El argumento pasado a una función matemática o de cadena estaba fuera del rango. Los errores FC pueden introducirse por: |

| CODIGO DE ERROR | SIGNIFICADO |
|---|---|
| | 1 Un subíndice negativo en matrices (LET A(−1) = 0).<br>2 Un subíndice excesivamente grande en matrices ( > 32767)<br>3 Argumento negativo ó 0 en un LOG.<br>4 Argumento negativo en un SQR.<br>5 A∧B con A negativo y B no entero.<br>6 Utilización USR sin haber almacenado adecuadamente la dirección de la subrutina en lenguaje máquina.<br>7 Argumento inadecuado en la utilización de MID$, LEFT$, RIGHT$, WAIT, PEEK, POKE, TAB, SPC ó ON . . . GOTO. |
| ID | Comando directo ilegal. Las instrucciones INPUT, DEF y GET no son utilizables como comandos directos. |
| LS | Cadena demasiado larga. Se intentó crear por encadenamiento una cadena de más de 255 caracteres. |
| NF | NEXT sin FOR. La variable en una instrucción NEXT no corresponde a una instrucción FOR previamente ejecutada. |
| OD | Datos insuficientes. Se ejecutó una instrucción READ cuando ya se habían leído todas las instrucciones DATA del programa. El programa intenta leer demasiados datos o incluye un número insuficiente de datos. |
| OM | Memoria insuficiente. Programa demasiado largo, demasiadas variables, demasiados lazos FOR, demasiados GOSUB, expresión demasiado complicada, o cualquier combinación de estos problemas. (Ver apéndice B). |
| OV | Desbordamiento. El resultado de un cálculo fue demasiado grande para representarse en el formato de los números en BASIC. Si el resultado desbordó el margen inferior (número demasiado pequeño), se |





| CODIGO DE ERROR | SIGNIFICADO |
|---|---|
| | tomará cero como resultado, la ejecución continuará y no se imprimirá ningún mensaje de error. |
| RG | RETURN sin GOSUB. Se encontró una instrucción RETURN sin que previamente se ejecutara una instrucción GOSUB. |
| SN | Error de sintáxis. Falta de paréntesis en una expresión, línea con carácter ilegal, puntuación incorrecta, etc. |
| ST | Cadenas temporales. Una expresión resultó demasiado compleja y se dividió en 2 ó más expresiones más cortas. |
| TM | Tipos no emparejados. Se intentó asignar a una variable numérica una cadena de caracteres o viceversa; o el argumento de una función, que debía ser una cadena de caracteres, encontró una variable numérica, o viceversa. |
| UF | Función indefinida. Se hizo referencia a una función que no había sido definida previamente. |
| US | Instrucción indefinida. Se intentó utilizar un GOTO, GOSUB, o THEN con destino inexistente. |
| /0 | División por cero. |

Las siguientes sugerencias pueden serle de utilidad para acortar sus programas y ahorrar espacio:

1. Use varias instrucciones en cada línea. Cada línea de programa lleva una cabecera de 5 bytes. Dos de estos bytes contienen el número de la línea en binario. O sea que independientemente del número que asigne a una línea (mínimo 0 máximo 63999), éste ocupará dos bytes de memoria. Incluyendo en una línea tantas instrucciones como quepan, usted ahorra bytes de memoria, pues evita los cinco bytes de encabezamiento.

2. Elimine de su programa todos los espacios innecesarios. Por ejemplo:

   10 PRINT X, Y, Z

   usa tres bytes más que

   10 PRINTX, Y, Z

   NOTA: Se ignoran todos los espacios entre el número de línea y el primer carácter significativo.

3. Elimine las instrucciones REM. Cada comentario REM usa al menos un byte más los utilizados en el texto. Por ejemplo, la línea 130 REM ESTO ES UN COMENTARIO usa 28 bytes de memoria. En la línea 140 X = X + Y: REM SUMA ACTUAL, el comentario ocupa 15 bytes de memoria, incluyendo los dos puntos anteriores al REM.

4. Use variables en vez de constantes. Supongamos qué su programa usa diez veces la constante 3.14159. Añadiendo a su programa un línea

   10 P=3.14159

   y usando cada vez que lo necesite P en vez de 3.14159, su programa ahorra 40 bytes. Esto también mejorará la velocidad.

5. Un programa no necesita terminar con END, así que puede eliminar la instrucción END del final de su programa.

6. Reutilice las variables. Si una parte de su programa utiliza una variable para almacenar temporalmente un resultado, y después necesita temporalmente una variable, vuelva a utililizar la anterior. O, si durante la ejecución de su programa pide al terminal del usuario un SIo NO a dos preguntas distintas en dos momentos distintos, almacene temporalmente la respuesta en la misma variable A$.

7. Utilice GOSUB para ejecutar secciones de programa que realicen la misma función.

8. Use el elemento cero de las matrices; por ejemplo A (0), B (0,X).





INFORMACION SOBRE ASIGNACION DE MEMORIA

Las variables numéricas o cadenas de caracteres, simples no matrices, como p.e. V, ocupan 7 bytes, 2 para el nombre de la variable, y 5 para el valor. Las variables simples de cadena, no matrices, también ocupan 7 bytes, 2 para el nombre de la variable, 1 para la longitud, 2 para un puntero, y 2 no se utilizan.

Las variables matriciales necesitan 7 bytes para almacenar el encabezamiento, más bytes adicionales para almacenar cada elemento de la matriz. Cada elemento que sea una variable de cadena de caracteres requiere 3 bytes, cada elemento que venga en coma flotante requiere 5 bytes, y cada elemento entero, 2 bytes.

Las variables de cadena también utilizan un byte por cada carácter en la cadena. Esto tanto en el caso de una variable simple de cadena como A$, o un elemento de una matriz de cadena como Q1$(5,2).

Al definir una función por medio de la instrucción DEF, se utilizan 7 bytes para almacenar la definición.

Las palabras reservadas, tales como FOR, GOTO, NOT y los nombres de las funciones intrínsecas, p.e. COS, INT, STR$, sólo necesitan un byte de almacenamiento. El resto de los caracteres en un programa utilizan un byte de almacenamiento por carácter.

Cuando se ejecuta un programa, se asigna dinámicamente espacio en el stack de la siguiente manera:

1. Cada ciclo FOR . . . NEXT activo usa 22 bytes.
2. Cada GOSUB activo (cuando aún no se ha regresado) usa 6 bytes.
3. Cada paréntesis de una expresión usa 4 bytes y cada resultado temporal calculado en una expresión usa 12 bytes.

## SECCION: APENDICES

## TEMA: SUGERENCIAS SOBRE VELOCIDAD

Las siguientes sugerencias pueden mejorar el tiempo de ejecución de sus programas BASIC. Observe que varias de estas sugerencias coinciden con las dadas para disminuir el espacio de almacenamiento del programa. Lo que significa que se puden mejorar simultáneamente la velocidad y el tamaño del programa.

1. Elimine todos los espacios innecesarios así como las instrucciones REM. Esto disminuirá ligeramente el tiempo de ejecución porque, en caso contrario, el BASIC tendría que ignorar o saltarse los espacios e instrucciones REM.

2. ESTA ES PROBABLEMENTE LA SUGERENCIA MAS IMPORTANTE PARA LA VELOCIDAD.
   Utilice variables en lugar de constantes. Es más lento convertir una constante a su representación en coma flotante, que ir a buscar el valor de una variable simple o matricial. Esto es especialmente importante en los ciclos FOR . . . NEXT y otros códigos de ejecución repetitiva.

3. Durante la ejecución de un programa BASIC, las variables son colocadas en la tabla de variables en orden de asignación. O sea que una línea tal como 5 A = 0: B = A:C = A, colocará en la tabla de variables primero A, segundo B y tercero C (suponiendo que la línea 5 era la primera ejecutada en que había asignaciones). Posteriormente cuando el BASIC encuentre una referencia a la variable A, para buscarca, tendrá que examinar sólo una posición de la tabla, dos posiciones para B, 3 para encontrar C, etc.

4. Use la instrucción NEXT sin la variable de índice. NEXT es algo más rápido que NEXT I porque no hay que comprobar si la variable I especificada en el NEXT corresponde con la variable en el último FOR.



## SECCION: APENDICES

## TEMA: CONVERSION DE PROGRAMAS BASIC NO ESCRITOS PARA EL BASIC DEL AIM 65

Aunque la implementación del lenguaje BASIC en diferentes computadores tiene muchos puntos comunes, hay algunas incompatibilidades que deben tenerse en cuenta si se intenta convertir algunos programas BASIC que no fueron escritos en el BASIC del AIM 65.

1. Notación matricial. Algunos BASIC usan "[" y "]" para indicar los subíndices de las matrices. El BASIC AIM 65 usa "(" y ")".

2. Cadenas de caracteres. Ciertos BASIC precisar declarar la longitud de las cadenas antes de utilizarlas. Todas las instrucciones de este tipo de dimensionado deberían eliminarse del programa. En algunas de estos BASIC, una instrucción DIM A$ (I, J) declara una matriz de cadenas de J, elementos, cada uno de ellos de una longitud I. Cambiar estas instrucciones DIM por la equivalente en el BASIC del AIM 65: DIM A$ (J).

El BASIC del AIM 65 utiliza " + " para la concatenación de cadenas, no "," o "&". El BASIS del AIM usa LEFT$, RIGHT$ y MID$ para subdividir una cadena. Otros BASIC utilizan A$ (I) para acceder al carácter número I de la cadena A$, y A$ (I, J) para tomar una subcadena de A$ desde el carácter número I al carácter número J. Convertirlas así:

| ANTERIOR | AIM 65 |
|---|---|
| A$ (I) | MID$(A$, I, 1) |
| A$ (I, J) | MID$(A$, I, J–I +1). |

Esto en el supuesto de que la subcadena de A$ se usase en una expresión o en el lado derecho de una asignación (o sea que A$ no se altera). Si la subcadena de A$ se usase en el lado izquierdo de una asignación y X$ es la cadena utilizada para cambiar caracteres en A$, entonces los cambios serían:

| ANTERIOR | AIM 65 |
|---|---|
| A$ (I) = X$ | A$ LEFT$ (A$, I–1) + X$ + MID$ (A$, I + 1) |
| A$ (I, J) = X$ | A$ LEFT$ (A$, I -1) + X$ + MID$ (A$, J + 1) |

3. Asignación múltiple. Algunos BASIC permiten instrucciones del tipo:

   500 LET B = C = 0. Esta instrucción asigna a B y C el valor cero.

   En el BASIC del AIM 65 el efecto sería totalmente diferente. Todos los " = " a la derecha del primero se tomarían por operadores de comparación lógica.

Esto haría que B = –1 si C es igual a 0 y B = 0 si C = 0. La manera más fácil de convertir instrucciones como esta es escribirlas así:

```
500  C=0 : B=C
```

4. Algunos BASIC usa "/" en vez de "." para separar las distintas instrucciones en una línea. Cámbiense todas las separaciones "/" por ":".

5. Los programas que usan la función MAT disponible en algunos BASIC, deben reescribirse usando ciclos FOR . . . NEXT para obtener la operación apropiada.

6. Una instrucción PRINT sin argumento no producirá un avance del papel de la impresora. Para avanzar el papel (línea en blanco), utilizar PRINT "espacio".





| DECIMAL | CHAR. | DECIMAL | CHAR. | DECIMAL | CHAR. |
|---|---|---|---|---|---|
| 000 | NUL | 040 | ( | 080 | P |
| 001 | SOH | 041 | ) | 081 | Q |
| 002 | STX | 042 | * | 082 | R |
| 003 | ETX | 043 | + | 083 | S |
| 004 | EOT | 044 | , | 084 | T |
| 005 | ENQ | 045 | - | 085 | U |
| 006 | ACK | 046 | . | 086 | V |
| 007 | BEL | 047 | / | 087 | W |
| 008 | BS | 048 | 0 | 088 | X |
| 009 | HT | 049 | 1 | 089 | Y |
| 010 | LF | 050 | 2 | 090 | Z |
| 011 | VT | 051 | 3 | 091 | [ |
| 012 | FF | 052 | 4 | 092 | / |
| 013 | CR | 053 | 5 | 093 | ] |
| 014 | SO | 054 | 6 | 094 | ↑ |
| 015 | SI | 055 | 7 | 095 | ← |
| 016 | DLE | 056 | 8 | 096 | ` |
| 017 | DC1 | 057 | 9 | 097 | a |
| 018 | DC2 | 058 | : | 098 | b |
| 019 | DC3 | 059 | ; | 099 | c |
| 020 | DC4 | 060 | < | 100 | d |
| 021 | NAK | 061 | = | 101 | e |
| 022 | SYN | 062 | > | 102 | f |
| 023 | ETB | 063 | ? | 103 | g |
| 024 | CAN | 064 | @ | 104 | h |
| 025 | EM | 065 | A | 105 | i |
| 026 | SUB | 066 | B | 106 | j |
| 027 | ESCAPE | 067 | C | 107 | k |
| 028 | FS | 068 | D | 108 | l |
| 029 | GS | 069 | E | 109 | m |
| 030 | RS | 070 | F | 110 | n |
| 031 | US | 071 | G | 111 | o |
| 032 | SPACE | 072 | H | 112 | p |
| 033 | ! | 073 | I | 113 | q |
| 034 | " | 074 | J | 114 | r |
| 035 | # | 075 | K | 115 | s |
| 036 | $ | 076 | L | 116 | t |
| 037 | % | 077 | M | 117 | u |
| 038 | & | 078 | N | 118 | v |
| 039 | ' | 079 | O | 119 | w |

| DECIMAL | CHAR. | DECIMAL | CHAR | DECIMAL | CHAR |
|---|---|---|---|---|---|
| 120 | x | 123 | { | 126 | ~ |
| 121 | y | 124 | \| | 127 | DEL |
| 122 | z | 125 | } | | |

Line Feed = Avance de Línea
Form Feed = Avande de Página
Carriage Return = Retroceso del Carro
Rubout = Borrar, tachar
on TTY = En teletipo





Hay situaciones en las que es mucho más eficaz programar alguna operación en ensamblador y no en BASIC. Para estas situaciones, el BASIC del AIM 65 tiene una instrucción especial que permite salir del programa BASIC, ejecutar un programa en lenguaje ensamblado (una subrutina), y después volver a la siguiente instrucción en el programa BASIC. Esta instrucción especial se llama USR(W).

LA INSTRUCCION USR(W)

Cuando se ejecuta una instrucción USR(W), el microprocesador del AIM 65 busca la dirección inicial de la subrutina en lenguaje ensamblado, en dos posiciones de memoria, 04 y 05, y comienza en esa dirección la ejecución de las instrucciones del programa. La última instrucción a realizar en la subrutina debe ser una instrucción RTS (vuelta desde subrutina), la cual devolverá el control al programa BASIC, en la instrucción siguiente a la USR(W).

El argumento W es un parámetro que puede pasarse del programa BASIC a la subrutina ensamblada. El BASIC tiene este parámetro en la memoria en un acumulador especial, en coma flotante. El programa ensamblado no puede utilizar el parámetro en coma flotante, pero usando la instrucción JSR BEFE se le convierte en un entero de 2 bytes con signo, contenido en las posiciones $AC (Byte más significativo) y $AD (Byte menos significativo). Se producirá un error FC si el parámetro es mayor que +32,767 o menor que −32,767.

Asimismo, antes de volver al BASIC, el entero de dos byte puede pasarse de nuevo a la forma flotante mediante la instrucción JSR C0D1. Previamente el byte más significativo debe estar en el acumulador y el menos significativo en el registro Y.

Se puede pasar más de un parámetro desde el BASIC a la subrutina. Pero se ha de usar la instrucción POKE en vez de USR(W). El argumento (0 a 255) de cada instrucción POKE será convertido automáticamente de coma flotante a entero de 1 byte y almacenado en la posición direccionada por la instrucción POKE.

ASIGNACION DE MEMORIA

El BASIC del AIM 65 puede utilizar toda o parte de la memoria de lectura/escritura, a partir de la posición 530, a no ser que se limite la cantidad de memoria al responder a la pregunta MEMORY SIZA?. Como se indica en la sección 100, si su AIM 65 tiene 4 K byte de RAM y Vd. responde con RETURN o 4096 al MEMORY SIZE?, las posiciones de memoria 530 a 4906, serán asignadas al BASIC.

Para incluir una subrutina en lenguaje ensamblador en los 4K del sistema, hay que dar una respuesta menor que 4096 a la pregunta MEMORY SIZE?. Por supuesto, la respuesta debe reflejar el número de posiciones de memoria que ocupa la subrutina. Se

contestaria con 4086 si la subrutina ocupa 10 bytes, ó 4046 si la subrutina ocupa 50 bytes.

INTRODUCCION DE LA SUBRUTINA EN MEMORIA

Normalmente se utilizará el modo de entrada de instrucciones mnemónicas del AIM 65 para almacenar la subrutina en memoria. La sección 3.5 de la Guia del Usuario describe este modo y la utilización del comando L

El modo de entrada de instrucciones mnemónicas opera con direcciones de memoria en hexadecimal. Así que si la respuesta a MEMORY SIZE? asigna 50 posiciones de memoria a la subrutina, ésta debería empezar en la dirección hexadecimal 0FCE en decimal (4095–49).

INTRODUCCION EN MEMORIA DE LA DIRECCION DE COMIENZO DE LA SUBRUTINA

Como se indicó anteriormente, la instrucción USR(W) usa el contenido de las posiciones 04 y 05 de memoria para determinar la dirección de comienzo de la subrutina en lenguaje ensamblador. Concretamente la posición 04 contiene el byte menos significativo de la dirección y la posición 05 el byte más significativo. ¿Cómo almacenar la dirección en estas posiciones?. Hay dos formas de hacerlo: Usando los comandos M y / del Monitor del AIM 65 (ver sección 3.4 de la Guia del Usuario del AIM 65) o usando el comando POKE del BASIC del AIM 65 (Ver sección 303 de este manual). ¿Cuál de estos caminos debe utilizarse?. Depende de la aplicación. El comando del Monitor permite almacenar en memoria la dirección hexadecimal, sin convertirla a decimal, pero no permite cambiar esta dirección sin parar el programa BASIC. El comando POKE admite cambios de dirección con el programa BASIC, lo que permite utilizar más de una subrutina en lenguaje ensamblador.

El formato del comando POKE es POKE dirección, byte, donde tanto la dirección como el byte deben expresarse en decimal. Esto obliga a usar dos comandos POKE para almacenar en memoria la dirección de comienzo de subrutina; el primer POKE almacena (en decimal) el LSB (byte menos significativo) en la posición 04, el segundo POKE almacena (en decimal) el MSB (byte más significativo) en la posición 05.

Por ejemplo, si la subrutina en lenguaje ensamblador comienza en la dirección hexadecimal 0FCE, hay que almacenar 206 en la posición 04 (206 en decimal = CE en hexadecimal) y 15 en la posición 05 (15 en decimal = 0E en hexadecimal). La instrucción para realizarlo será:

```
10 POKE 04, 206: POKE 05,15
```





UN EJEMPLO DE INSTRUCCION USR(W)

Veamos un ejemplo real de utilización de la instrucción USR(W) con una subrutina en lenguaje ensamblador. El ejemplo tomará del teclado un número de cuatro dígitos hexadecimales, lo convertirá a decimal e imprimirá el resultado. Esta rutina de conversión sería útil para convertir direcciones hexadecimales a decimales, por lo que puede usarse con comandos PEEK y POKE. El número hexadecimal a convertir debe de estar comprendido entre 0000 y 7FFF.

La figura F–1 muestra el programa BASIC que llama a la subrutina de conversión hexadecimal. En este programa se limita la memoria del BASIC desde la $0212 hasta la $0FA0 (4000 en decimal) contestando 4000 a la pregunta MEMORY SIZE?, dejando libre las posiciones $0FA1 y siguientes para aceptar programas en lenguaje ensamblador. En la línea 5 se especifica por medio de dos comandos POKE que la subrutina de conversión comienza en la posición $0FA1. La línea 15 tome del teclado una cadena de cuatro dígitos y la almacena en la matriz H. El bucle formado por las líneas 20, 25, 30 y 35, convierten cada elemento de la matriz a un código ASCII y el comando POKE que los guarda sucesivamente en cuatro posiciones de memoria, empezando en la dirección 4049 en decimal ($0FD1). Cuando los cuatro dígitos se han procesado, el BASIC llama a la subrutina de conversión, con la instrucción USR(I), y a la vuelta imprime el número decimal equivalente. La instrucción GOTO 15 de la línea 45 hace que el BASIC vuelva a la línea 15, para aceptar un nuevo número hexadecimal.

En la figura F–2 puede verse la subrutina de conversión, en lenguaje ensamblado. Las cuatro primeras instrucciones buscan los códigos ASCII de los dos dígitos menos significativos (en hexadecimal) y, por medio de la subrutina PACK del Monitor (dirección de entrada $EA84), los convierte en dígitos en BCD (Binary – Coded – Decimal) y los empaqueta en el acumulador. La instrucción siguiente. TAY, transfiere los dos dígitos BCD empaquetados al registro Y.

Las cuatro instrucciones siguiente hacen dos nuevas llamadas a PACK, para los dos dígitos más significativos, dejando el resultado en el acumulador. Ahora, la instrucción JSR/C0D1 convierte el número de cuatro dígitos BCD a coma flotante, la instrucción RTS devuelve el control al programa BASIC. En la figura F–3 puede verse la impresión de resultados al ejecutar 6 veces el programa BASIC.

```
(5)
MEMORY SIZE? 4000
WIDTH?
3470 BYTES FREE
 AIM 65 BASIC V1.1
 5 POKE4,161:POKE5,15
```

```
10 DIM H(4)
15 INPUT H$
20 FOR I = 1TO 4
25 H(I) = ASC(MID$ (H$, I,1))
30 POKE 4048 + I,H(I)
35 NEXT
40 X = USR(I):PRINT X
45 GOTO15
```

Fig. F-1. Un programa BASIC que llama a una subrutina en lenguaje ensamblado.

```
(*)=0FA1
(I)
0FA1  AD    LDA   0FD3
0FA4  20    JSR   EA84
0FA7  AD    LDA   0FD4
0FAA  20    JSR   EA84
0FAD  A8    TAY
0FAE  AD    LDA   0FD1
0FB1  20    JSR   EA84
0FB4  AD    LDA   0FD2
0FB7  20    JSR   EA84
0FBA  20    JSR   C0D1
0FBD  60    RTS
0FBE
```

Fig. F-2. Una subrutina de conversión de 5 dígitos hexadecimales a decimal.

```
(6)
RUN
? 0001
  1
? 000A
  10
? 00A0
  160
? 0A00
  2560
? 0FA1
  4001
? 0FD1
  4049
```

Fig. F-3. Impresión de resultados de seis conversiones hexadecimales.





Los programas en BASIC del AIM 65 pueden guardarse en cintas de cassette usando los comandos SAVE y LOAD del BASIC. Asegurarse de seguir atentamente las indicaciones de instalación y operación de la sección 9 de la Guía del Usuario del AIM 65.

## GRABACION EN CASSETTE MEDIANTE EL COMANDO SAVE DEL BASIC

El proceso de grabación de un programa BASIC es:

1. Cambiar el valor de la posición de memoria $A409, con las siguientes consideraciones:

   a) Si la recuperación del programa se hace sólo mediante control manual del cassette, almacenar $20 en la posición de memoria $A409.

   b) Si la recuperación del programa se hace con control remoto, el valor a almacenar en la posición $A409 dependerá de las características del motor del magnetófono utilizado. El valor estará comprendido entre $30 (para la mayoría de los magnetófonos General Electric) y $80 (para la mayoría de loa magnetófonos Radio Shack). Para mayor información ver Sección 9.1.5 de la Guía del Usuario del AIM 65.

   c) Si la recuperación del programa se hace con un magnetófono del que no se conocen las características, almacenar $80 en la posición $A409.

2. Instalar un cassette en el magnetófono, inicializar el contador al principio de la cinta y situar, con control manual, la cinta en la posición donde quiera realizar la grabación.

3. Estando en BASIC teclear SAVE, el BASIC contestará

   OUT?

4. Teclear T (de Tape, para salida a magnetófono), el BASIC contestará

   OUT = T  F =

5. Teclear el nombre del fichero (hasta 5 caracteres; si son menos de 5 terminar con RETURN) si el nombre el FNOMB, el BASIC contestará

   OUT = T  F = FNOMB  T =

6. Poner el magnetófono en grabación.

7. Teclear el número del magnetófono (1 ó 2) y RETURN.

8. Si se utiliza el control remoto, deben seguirse las indicaciones de la sección 9.1.5. de la Guía del Usuario del AIM 65.

9. Al terminarse la grabación, el BASIC presentará el cursor en el display. Interrumpir la grabación del magnetófono.

RECUPERACION DE UN PROGRAMA EN CASSETTE MEDIANTE EL COMANDO LOAD DEL BASIC

El proceso de recuperación de un programa BASIC es:

1. Instalar el cassette en el magnetófono, situar la cinta con control manual aproximadamente cinco cuentas antes del principio del fichero a recuperar.

   NOTA: Si la impresora está activada, se listará el programa BASIC según se carga. Se necesitará control remoto para cargar un programa con la impresora activa. De producirse errores de lectura durante la operación de carga, desactive la impresora (con CTRL PRINT y reinicie la carga.

2. Estando en BASIC, teclear LOAD. El BASIC responderá:

   IN =

3. Teclear una T (de Tape, cinta). El BASIC contestará:

   IN = T F =

4. Teclear el nombre del fichero, si éste es FNOMB el BASIC contestará:

   IN = T F = FNOMB T =

5. Teclear el número del magnetófono (1 ó 2) y RETURN.

6. Poner el magnetófono en reproducción. Asegurarse de seguir las indicaciones de la sección 9.1.6 de la Guía del Usuario del AIM 65.

   Mientras se lee el fichero, se presentará cada línea (y se imprimirá si la impresora está activa). Si la impresora está activa, probablemente sea necesario aumentar el espacio entre registro de la cinta ($A409).



SECCION: APENDICES

TEMA: GRABACION DE PROGRAMAS BASIC AIM 65 EN CASSETTE

   La lectura del fichero no recubrirá instrucciones BASIC anteriores salvo que los números de sus líneas coincidan.

7. Al término dela carga el BASIC presentará el cursor.

8. Interrumpir la reproducción del magnetófono.

La función ATN (Ver tema 307) puede programarse en RAM usando el modo de entrada nemónica y el comando de alteración de posiciones de memoria, tal como se muestra a continuación. Este programa está escrito para el AIM 65 con 4 k bytes de RAM. La función ATN puede reubicarse en cualquier posición de memoria cambiando la dirección de comienzo de instrucciones y constantes, las direcciones de los saltos condicionales, el vector a la dirección de comienzo de constantes y el vector a la dirección de comienzo de la función ATN.

CONSTANTES DE LA FUNCION ATN ALMACENADA MEDIANTE EL COMANDO M DE ALTERACION DE MEMORIA

```
M   0F80  XX  XX  XX  XX   Dirección de Comienzo de Constantes
                           0F80
/   0F80  0B  76  B3  83
/   0F84  BD  D3  79  1E
/   0F88  F4  A6  F5  7B
/   0F8C  83  FC  80  10
/   0F90  7C  0C  1F  67
/   0F94  CA  7C  DE  53
/   0F98  CB  C1  7D  14
/   0F9C  64  70  4C  7D
/   0FA0  B7  EA  51  7A
/   0FA4  7D  63  30  88
/   0FA8  7E  7E  92  44
/   0FAC  99  3A  7E  4C
/   0FB0  CC  91  C7  7F
/   0FB4  AA  AA  AA  13
/   0FB8  81  00  00  00
/   0FBC  00
```

INSTRUCCIONES DE LA FUNCION ATN ALMACENADAS MEDIANTE EL COMANDO (I) DE ENTRADA DE NEMONICOS

```
I                          Dirección de Comienzo de Instrucciones
                           0FBD

XXXX* = 0FBD
0FBD  A5  LDA  AE
0FBF  48  PHA
0FC0  10  BPL  0FC5
0FC2  20  JSR  CCB8
0FC5  A5  LDA  A9
0FC7  48  PHA
0FC8  C9  CMP  #81
0FCA  90  BCC  0FD3
0FCC  A9  LDA  #FB
0FCE  A0  LDY  #C6
0FDO  20  JSR  C84E
```





```
0FD3  A9  LDA  #80   Dirección de Comienzo de Constantes
0FD5  A0  LDY  #0F   0F80

0FD7  20  JSR  CD44
0FDA  68  PLA
0FDB  C9  CMP  #81
0FDD  90  BCC  0FE6
0FDF  A9  LDA  #4E
0FE1  A0  LDY  #CE
0FE3  20  JSR  C58F
0FE6  68  PLA
0FE7  10  BPL  0FEC
0FE0  4C  JMP  CCB8
0FEC  60  RTS
```

INICIALIZACION DEL BASIC PARA LA FUNCION ATN

La memoria BASIC debe inicializarse, después de la asignación de memoria de la función ATN. También debe cambiarse el vector de ATN en RAM de la dirección del mensaje de error FC a la dirección inicial de las instrucciones de la Función ATN.

Esto puede hacerse usando la inicialización BASIC así:

```
 5
MEMORY SIZE? 3958
AIM 65 BASIC V1.1
POKE 188,189          Cambiar LSB del vector ATN a $BD=189
POKE 189,15           Cambiar MSB del vector ATN a $0F=15
?ATN(TAN(.5))         Prueba de funcionamiento del programa ATN
.5                    Respuesta esperada .5
```

GRABACION DEL CODIGO OBJETO DE ATN EN CASSETTE

El código objeto de la función ATN puede guardarse en cassette volcando (D) las direcciones $00BD (Instrucciones de salto a ATN) y $0F80 a $FEC (instrucciones y contantes), después de la carga inicial de la función ATN y su verificación.

La recuperación desde un cassette de la función ATN se hará ejecutando el comando L del monitor después de inicializar el BASIC con el comando 5. Después de cargar la función ATN se vuelve al BASIC con el comando 6.